# AUTHOR INDEX

Abdali . . . 253 3/F10
Ament . . . 87 2/A7
Andersen . . . 161 2/F11
Anderson . . . 395 5/C8
Avgoustis . . . 21 1/C6
Bogen . . . 11, 75 1/B10
Bulnes . . . 461 6/A9
Caviness . . . 253 3/F10
Char . . . 53 1/E10
Cohen . . . 275 4/A9
Cuthill . . . 131 2/E3
Doohovskoy . . . 473 6/B7
Fateman . . . 43, 327 1/D14
Fennelly . . . 97 2/B3
Geddes . . . 405 5/D4
Genesereth . . . 291, 309 4/B11
Golden . . . 1, 109 1/A14
Gosper . . . 237 3/E8
Grünbaum . . . 491 6/C11
Gupta . . . 151 2/F1
Ivie . . . 317 4/D9
Karney . . . 377 5/B4
Kulp . . . 385 5/B12
Lafferty . . . 347 4/F11
Lau . . . 395 5/C8
Lewis . . . 277 4/A11
Moenck . . . 225 3/D10
Moses . . . 123, 275 2/D1
Ng . . . 151, 177 2/F1
Noor . . . 161 2/F11
Pavelle . . . 75, 97 1/G4
Poole . . . 145 2/E9
Pridor . . . 253 3/F10
Rothstein . . . 263 3/G6
Spear . . . 369 5/A10
Steele . . . 203, 215 3/C2
Stoutemyer . . . 315, 425, 447 4/D7
Wahlquist . . . 71 1/F14
Wang . . . 55, 435 1/E12
Yun . . . 65 1/F8
Zippel . . . 361 4/G11

# CONTENTS

FOREWORD . . . . . . . . . . . . . . . . . . . . . . . . . . . . iii 1/A10

PREFACE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . v 1/A12

AUTHOR INDEX . . . . . . . . . . . . . . . . . . . . . . . . . . vii 1/A1

1. MACSYMA'S SYMBOLIC ORDINARY DIFFERENTIAL EQUATION SOLVER . . . . . . . 1 1/A14
   Jeffrey P. Golden

2. A PROGRAM FOR THE SOLUTION OF INTEGRAL EQUATIONS . . . . . . . . . . . 11 1/B10
   Richard A. Bogen

3. SYMBOLIC LAPLACE TRANSFORMS OF SPECIAL FUNCTIONS . . . . . . . . . . . 21 1/C6
   Yannis Avgoustis

4. AN IMPROVED ALGORITHM FOR THE ISOLATION OF POLYNOMIAL REAL ZEROS . . . 43 1/D14
   Richard J. Fateman

5. FLOATING POINT ISOLATION OF ZEROS OF A REAL POLYNOMIAL VIA MACSYMA . . 53 1/E10
   Bruce W. Char

6. PRESERVING SPARSENESS IN MULTIVARIATE POLYNOMIAL FACTORIZATION . . . . 55 1/E12
   Paul S. Wang

7. ON THE EQUIVALENCE OF POLYNOMIAL GCD AND SQUAREFREE FACTORIZATION PROBLEMS . . . . . . . . . . . . . . . . . . . . . . . . . . 65 1/F8
   David Y. Y. Yun

8. DIFFERENTIAL FORM ANALYSIS USING MACSYMA . . . . . . . . . . . . . . . 71 1/F14
   Hugo D. Wahlquist

9. INDICIAL TENSOR MANIPULATION ON MACSYMA . . . . . . . . . . . . . . . 75 1/G4
   Richard A. Bogen and Richard Pavelle

10. PURE FIELD THEORIES AND MACSYMA ALGORITHMS . . . . . . . . . . . . . 87 2/A7
    William S. Ament

11. BLACK HOLES AND RELATIVISTIC GRAVITY THEORIES . . . . . . . . . . . . 97 2/B3
    A. J. Fennelly and Richard Pavelle

12. THE EVALUATION OF ATOMIC VARIABLES IN MACSYMA . . . . . . . . . . . . 109 2/C1
    Jeffrey P. Golden

13. THE VARIETY OF VARIABLES IN MATHEMATICAL EXPRESSIONS . . . . . . . . 123 2/D1
Joel Moses

14. RATIONAL APPROXIMATION TO $e^{-x}$ WITH NEGATIVE REAL POLES . . . . . . . 131 2/D9
Elizabeth Cuthill

15. TIMING FORMULAS FOR DISSECTION ALGORITHMS ON VECTOR COMPUTERS . . . . 145 2/E9
W. G. Poole, Jr.

16. SYMBOLIC CALCULATIONS IN A FINITE DYNAMIC ELEMENT ANALYSIS . . . . . 151 2/F1
Kajal K. Gupta and Edward W. Ng

17. SYMBOLIC MANIPULATION TECHNIQUES FOR VIBRATION ANALYSIS OF LAMINATED ELLIPTIC PLATES . . . . . . . . . . . . . . . . . . . . 161 2/F11
C. M. Andersen and Ahmed K. Noor

18. OBSERVATIONS ON APPROXIMATE INTEGRATIONS . . . . . . . . . . . . . . . 177 2/G13
Edward W. Ng

19. LISP: PROGRAM IS DATA — A HISTORICAL PERSPECTIVE ON MACLISP . . . . 181 3/A8
Jon L White

20. LISP: DATA IS PROGRAM — A TUTORIAL IN LISP . . . . . . . . . . . . . 191 3/B4
Jon L White

21. DATA REPRESENTATIONS IN PDP-10 MACLISP . . . . . . . . . . . . . . . . 203 3/C2
Guy Lewis Steele Jr.

22. FAST ARITHMETIC IN MACLISP . . . . . . . . . . . . . . . . . . . . . 215 3/C14
Guy Lewis Steele Jr.

23. ON COMPUTING CLOSED FORMS FOR SUMMATIONS . . . . . . . . . . . . . . 225 3/D10
Robert Moenck

24. INDEFINITE HYPERGEOMETRIC SUMS IN MACSYMA . . . . . . . . . . . . . . 237 3/E8
R. Wm. Gosper, Jr.

25. MODULAR POLYNOMIAL ARITHMETIC IN PARTIAL FRACTION DECOMPOSITION . . . 253 3/F10
S. K. Abdali, B. F. Caviness, and A. Pridor

26. A NEW ALGORITHM FOR THE INTEGRATION OF EXPONENTIAL AND LOGARITHMIC FUNCTIONS . . . . . . . . . . . . . . . . . . . . . . . 263 3/G6
Michael Rothstein

27. SUMMATION OF RATIONAL EXPONENTIAL EXPRESSIONS IN CLOSED FORM . . . . 275 4/A9
Joel Moses and Jacques Cohen

28. USER AIDS FOR MACSYMA . . . . . . . . . . . . . . . . . . . . . . . . 277 4/A11
V. Ellen Lewis

29. THE DIFFICULTIES OF USING MACSYMA AND THE FUNCTION OF USER AIDS . . . 291 4/B11
Michael R. Genesereth

30. AN AUTOMATED CONSULTANT FOR MACSYMA . . . . . . . . . . . . . . . . . . . . 309 4/D1
Michael R. Genesereth

31. A MACSYMA COMPUTER-ALGEBRA MOVIE DEMONSTRATION . . . . . . . . . . . 315 4/D7
David R. Stoutemyer

32. SOME MACSYMA PROGRAMS FOR SOLVING DIFFERENCE EQUATIONS . . . . . . . 317 4/D9
John Ivie

33. SOME COMMENTS ON SERIES SOLUTIONS . . . . . . . . . . . . . . . . . . . . . 327 4/E5
Richard J. Fateman

34. POWER SERIES SOLUTIONS OF ORDINARY DIFFERENTIAL EQUATIONS IN MACSYMA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 347 4/F11
Edward L. Lafferty

35. RADICAL SIMPLIFICATION MADE EASY . . . . . . . . . . . . . . . . . . . . 361 4/G11
Richard E. B. Zippel

36. A CONSTRUCTIVE APPROACH TO COMMUTATIVE RING THEORY . . . . . . . . . 369 5/A10
David A. Spear

37. REDUCTION OF THE EQUATION FOR LOWER HYBRID WAVES IN A PLASMA TO A NONLINEAR SCHRÖDINGER EQUATION . . . . . . . . . . . . . . . . . 377 5/B4
Charles F. F. Karney

38. RAY TRAJECTORIES IN A TORUS: AN APPLICATION OF MACSYMA TO A COMPLEX NUMERICAL COMPUTATION . . . . . . . . . . . . . . . . . . . . . . . 385 5/B12
John L. Kulp

39. APPLICATION OF MACSYMA TO FIRST ORDER PERTURBATION THEORY IN CELESTIAL MECHANICS . . . . . . . . . . . . . . . . . . . . . . . . . . 395 5/C8
John D. Anderson and Eunice L. Lau

40. SYMBOLIC COMPUTATION OF RECURRENCE EQUATIONS FOR THE CHEBYSHEV SERIES SOLUTION OF LINEAR ODE's . . . . . . . . . . . . . . . . . . . 405 5/D4
K. O. Geddes

41. sin(x)**2 + cos(x)**2 = 1 . . . . . . . . . . . . . . . . . . . . . . . 425 5/E10
David R. Stoutemyer

42. MATRIX COMPUTATIONS IN MACSYMA . . . . . . . . . . . . . . . . . . . . . 435 5/F6
Paul S. Wang

43. SYMBOLIC COMPUTER VECTOR ANALYSIS . . . . . . . . . . . . . . . . . . 447 5/G4
David R. Stoutemyer

44. A NATURAL WAY TO DO SPATIAL LINEAR GEOMETRY IN MACSYMA . . . . . . . 461 6/A9
Juan Bulnes

45. VARIETIES OF OPERATOR MANIPULATION . . . . . . . . . . . . . . . . . 473 6/B7
Alexander Doohovskoy

46. PROGRESS REPORT ON THE DETERMINANT OF A RANDOM MATRIX . . . . . . . . 491 6/C11
F. A. Grünbaum

BIBLIOGRAPHY . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 493 6/C13

AUG 10 1977

NASA CP 2012

NAS 1.55:2012

COMPLETED

ORIGINAL

# Proceedings of the 1977 MACSYMA Users' Conference

Held at
University of California
Berkeley, California
July 27-29, 1977

NASA

NASA Conference Publications (CP Series) contain compilations of scientific and technical papers or transcripts arising from conferences, workshops, symposia, seminars, and other professional meetings that NASA elects to publish.

The text of these proceedings was reproduced directly from author-supplied manuscripts for distribution prior to the 1977 MACSYMA Users' Conference. Only minimal editing was performed by NASA in order to preserve the informal style and vernacular of MACSYMA users.

NAS 1.55:2012

NASA CP-2012

# Proceedings of the 1977 MACSYMA Users' Conference

Sponsored by

Massachusetts Institute of Technology,
University of California at Berkeley,
NASA Langley Research Center

and held at Berkeley, California
July 27-29, 1977

NASA

Scientific and Technical Information Office 1977
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
Washington, D.C.

For sale by the National Technical Information Service
Springfield, Virginia 22161
Price: $12.50

# FOREWORD

The technical program of the 1977 MACSYMA Users' Conference, held at Berkeley, California, from July 27 to July 29, 1977, consisted of the 45 contributed papers reported in this publication and of a workshop. The workshop was designed to promote an exchange of information between implementers and users of the MACSYMA computer system and to help guide future developments.

The response to the call for papers has well exceeded the early estimates of the conference organizers; and the high quality and broad range of topics of the papers submitted has been most satisfying. A bibliography of papers concerned with the MACSYMA system is included at the end of this publication.

We would like to thank the members of the program committee, the many referees, and the secretarial and technical staffs at the University of California at Berkeley and at the Laboratory for Computer Science, Massachusetts Institute of Technology, for shepherding the many papers through the submission-to-publication process. We are especially appreciative of the burden carried by V. Ellen Lewis of M.I.T. for serving as expert in document preparation from computer-readable to camera-ready copy for several papers.

This conference originated as the result of an organizing session called by Joel Moses of M.I.T. at the 1976 ACM Symposium on Symbolic and Algebraic Computation, at Yorktown Heights, New York, in August 1976. It owes its success to his continuing encouragements and efforts, not to mention his intellectual and practical skills in keeping the MACSYMA project thriving.

We wish to acknowledge the kind cooperation of ACM, ACM-SIGSAM, the Electronics Research Laboratory and the Department of Electrical Engineering and Computer Sciences of the University of California, the Laboratory for Computer Science of M.I.T., NASA Langley Research Center, and the U.S. Energy Research and Development Administration.

We wish to extend our gratitude to the Scientific and Technical Information Programs Division of the NASA Langley Research Center for publishing these proceedings.

Richard J. Fateman, General Chairman

Carl M. Andersen, Program Committee Chairman

## OFFICERS OF THE 1977 MACSYMA USERS' CONFERENCE

General Chairman:

Richard Fateman, University of California, Berkeley

Program Chairman:

Carl M. Andersen, The College of William and Mary in Virginia

Program Committee:

Mary Ellen Brennan, Aerospace Corporation, Los Angeles

Jo Ann Howell, Los Alamos Scientific Laboratory, University of California

John Kulp, Research Laboratory of Electronics and Plasma Fusion Center, Massachusetts Institute of Technology

Joel Moses, Laboratory for Computer Science, Massachusetts Institute of Technology

Edward Ng, Jet Propulsion Laboratory, California Institute of Technology

David Stoutemyer, University of Hawaii

Editorial Committee:

Carl M. Andersen, The College of William and Mary in Virginia

Jeffrey Golden, Laboratory for Computer Science, Massachusetts Institute of Technology

John N. Shoosmith, NASA Langley Research Center

Treasurer:

Michael Genesereth, Harvard University and Massachusetts Institute of Technology

Local Arrangements:

Richard Fateman, University of California, Berkeley

# PREFACE

Symbolic and algebraic manipulation enables one to do exact, symbolic mathematical computations on a computer. To illustrate the difference between numeric and symbolic processing, consider a computer program (in FORTRAN, say) which, given the quantities A, B, and C, can apply the quadratic formula to approximate the roots of the quadratic equation A*x**2+B*x+C = 0. The names A, B, and C, must of course correspond to numerical values at run-time. This is because the program has been written to provide numerical processing. If A had as its run-time value the expression "Q," B had value "(-P*Q-1)," and C had value "P," the FORTRAN program would be useless. Nevertheless, by applying the quadratic formula symbolically, the two roots [-(-P*Q-1)±SQRT(P**2*Q**2+2*P*Q+1-4*P*Q)]/(2*Q) can be represented. By further efforts, this expression can be reduced to the set of values (P, 1/Q). This substitution (in this case, into the quadratic formula) and subsequent simplification are but two of the necessary operations in an algebra system. Some of the more elaborate facilities that can be built up (and have been, in MACSYMA) include partial differentiation, indefinite integration, inversion of matrices with symbolic coefficients, solution of polynomial equations, and manipulation of truncated power series. The range of capabilities can be seen in the papers in this conference.

MACSYMA is a large symbolic and algebraic manipulation system which has been under development at the Laboratory for Computer Science (formerly Project MAC) of the Massachusetts Institute of Technology since 1969. The system has more than quintupled in size since the first paper describing it appeared in 1971. It is, by any measure, a rather large program, and this makes it a challenging project from many points along the computer hardware-software spectrum. Some papers on the LISP system in these proceedings address this issue.

During the last several years, the community of users of the MACSYMA system has grown at an increasing rate; and because of the wide geographical range of the ARPA computer communication network of the Defense Communication Agency, there are now users from Hawaii to Cambridge, England. Another contributing factor in the growth has been the ability of Joel Moses and his staff at the Laboratory for Computer Science to make available at relatively low cost the most versatile of algebraic manipulation systems currently implemented. Another is the synergistic effect of the community itself: where the output of one person's program may be the input to the next person's, and where nearly instantaneous feedback on features and repair of bugs are the rule rather than the exception.

Many of the users of MACSYMA (including contributors to this conference) are also using or have used other systems (ALTRAN, FORMAC, REDUCE, SAC-1, and SCRATCHPAD, to name a few) with symbolic and algebraic manipulation facilities. Many of the techniques are not specific to MACSYMA, but are algebraic manipulation contributions independent of particular system context. Thus we view this conference as a collection of persons interested in advancing the field of inquiry in "symbolic and algebraic manipulation," and applying the fruits of this inquiry to other areas. We believe the papers bear out this view.

Until recently, major funding for MACSYMA development has come from the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract N00014-70-0362-0006. More recent additions to the sponsors' ranks have come from agencies whose own personnel and contractors have used MACSYMA. These include the U.S. Energy Research and Development Administration, the National Aeronautics and Space Administration, and the U.S. Navy. Combining resources to provide the unique facility of the MACSYMA Consortium, these sponsors have provided an invaluable resource.

Richard J. Fateman
General Chairman

# MACSYMA's Symbolic Ordinary Differential Equation Solver [*]

Jeffrey P. Golden
Laboratory for Computer Science
Massachusetts Institute of Technology

## ABSTRACT

This paper describes MACSYMA's symbolic ordinary differential equation solver ODE2. Although available in MACSYMA for approximately three years now, a paper describing how to use it had never previously been written. Also, this paper showcases the code for this routine, which is of interest because it is written in top-level MACSYMA language, and may serve as a good example of programming in that language. Other symbolic ordinary differential equation solvers are mentioned.

## 1. The ODE2 Package

MACSYMA's ordinary differential equation (ODE) solver ODE2 may be used for symbolically solving elementary ODEs of first and second order. It consists primarily of a set of routines based on techniques described in reference 1 for Moses' SOLDIER ODE program, and in reference 2, which had been used until recently as the major textbook in M.I.T.'s introductory ODE course 18.03. The ODE2 package was written primarily by an M.I.T. graduate student, Ben Kuipers, as a term project in a seminar on algebraic manipulation taught by Richard Fateman in the fall of 1972-73. It has since been maintained, modified, and improved by the author.

When the user calls the ODE2 routine, e.g. as follows:

```
(C1) X^2*'DIFF(Y,X) + 3*X*Y = SIN(X)/X;
                              2 dY            SIN(X)
(D1)                         X  -- + 3 X Y = ------
                                dX              X

(C2) ODE2(%,Y,X);
```

---

[*] This work was supported, in part, by the United States Energy Research and Development Administration under Contract Number E(11-1)-3070 and by the National Aeronautics and Space Administration under Grant NSG 1323.

the ODE package ODER LISP DSK SHARE (or ODER FASL DSK SHARE if the user is using NEWIO MACSYMA) is automatically loaded in. Or, the user can load it in by typing e.g. `LOADFILE(ODER,LISP,DSK,SHARE);`. For this example, after several out-of-core files are loaded in, the answer is obtained:

```
                                        C - COS(X)
(D2)                               Y = ----------
                                             3
                                            X
```

We see from this example how ODE2 is used. Namely, it takes three arguments: an ODE of first or second order (only the left hand side need be given if the right hand side is 0), the dependent variable, and the independent variable. When successful, it returns either an explicit or implicit solution for the dependent variable. C is used to represent the constant in the case of first order equations, and K1 and K2 the constants for second order equations. An alternative scheme, which has been suggested, of generating sequences of constants, e.g. K1, K2, K3, ..., so that different solutions might use different "constants", has not yet been implemented. If ODE2 cannot obtain a solution for whatever reason, it returns FALSE, after perhaps printing out an error message to the user.

The methods implemented for first order equations in the order in which they are tested are: linear, exact - perhaps requiring an integrating factor, homogeneous, Bernoulli's equation, and a generalized homogeneous method described in reference l.

For second order: constant coefficient, exact, linear homogeneous with non-constant coefficients which can be transformed to constant coefficient, the Euler or equidimensional equation, the method of variation of parameters, and equations which are free of either the independent or of the dependent variable so that they can be reduced to two first order linear equations to be solved sequentially.

In the course of solving ODEs, several variables are set purely for informational purposes: METHOD denotes the method of solution used e.g. LINEAR, INTFACTOR denotes any integrating factor used, ODEINDEX denotes the index for Bernoulli's method or for the generalized homogeneous method, and YP denotes the particular solution for the variation of parameters technique.

Since the code is written in top-level MACSYMA language, it may easily be extended not only by the author, but by other MACSYMA users as well. Indeed, there is much room for extension and improvement. The basic approach used in ODE2 is a pattern-directed one relying heavily on the MACSYMA commands EXPAND, COEFF, FREEOF, DERIVDEGREE, HIPOW, and SUBST, and on the MACSYMA pattern matcher DEFMATCH in checking for linearity. The basic power of the routine comes from MACSYMA's advanced indefinite integration package (ref. 3) and, of course, the INTEGRATE command is heavily used. Finally, basic restructuring of expressions is needed throughout, and for this RATSIMP is used heavily.

In order to solve initial value problems (IVPs) and boundary value problems (BVPs), the routine IC1 is available for first order equations, and IC2 and BC2 written by David Stoutemyer for second order IVPs and BVPs, respectively. They are used as in the following examples:

```
(C3) IC1(D2,X=%PI,Y=0);
                                 COS(X) + 1
(D3)                       Y = - ----------
                                      3
                                     X

(C4) 'DIFF(Y,X,2) + Y*'DIFF(Y,X)^3 = 0;
                           2
                          d Y       dY 3
(D4)                      --- + Y (--)  = 0
                            2       dX
                          dX

(C5) ODE2(%,Y,X);
                         3
                        Y  - 6 K1 Y - 6 X
(D7)                    ----------------- = K2
                                3

(C8) RATSIMP(IC2(D7,X=0,Y=0,'DIFF(Y,X)=2));
                             3
                          2 Y  - 3 Y + 6 X
(D9)                    - ---------------- = 0
                                 3

(C10) BC2(D7,X=0,Y=1,X=1,Y=3);
                         3
                        Y  - 10 Y - 6 X
(D11)                   --------------- = - 3
                               3
```

(The jumps in the line-number in the above examples are due to "hidden" calls to SOLVE.)

In order to see more clearly which methods have been implemented, a demonstration file is available. To run it, the user may do DEMO(ODER,DEMO,DSK,SHARE); and follow the usual prescription for running DEMO files as noted in the MACSYMA Manual (ref. 4).

The ODE2 package was used heavily in the work described by Richard Fateman in reference 5, in David Stoutemyer's OPTVAR variational optimization package, available via the SHARE file directory and described in reference 6, and in Stoutemyer's INTEQN integral equation solver, implemented in MACSYMA by Richard Bogen, also available via the SHARE directory and described in reference 7.

## 2. Other Symbolic ODE Solvers

Another program for solving ODEs which uses a heuristic search approach, and is called EULE, is described in references 8,9. Its author, Peter Schmidt of the University of Bonn, West Germany, did not have access to a powerful algebraic manipulation system and integration package such as with MACSYMA, so he was forced to implement his own simplification routines and EULE does not solve the integrals generated in its solutions. EULE solves only ODEs of the first order. However, Schmidt claims a high success rate in this area. EULE does handle a few more first order cases than ODE2 currently does, e.g. Riccati equations, and EULE's heuristic techniques may enable it to solve some "interesting" ODEs; however, the author believes that ODE2 could handle all of these cases as well with at most a few more pages of MACSYMA code. In fact, since the simplification and transformation capabilities of MACSYMA are so much more powerful than those of EULE, in experiments run by the author it turned out that several ODEs which Schmidt claims required heuristics and substitutions of variables in EULE, were actually solvable in ODE2 by more elementary methods, e.g. integrating factors or the generalized homogeneous method (which is not used by EULE as such.) ODE2 is much more successful than EULE in using methods that are implemented in both. (It is interesting to note that ODE2's first order methods, while not nearly as extensive as EULE's, only amount to 70 lines of MACSYMA code. Of course, ODE2 has some second order methods as well, and these amount to 120 lines of MACSYMA code. I think this data offers an interesting measure of the power of MACSYMA! EULE which together with all of its components has been developed only for the purpose of solving ODEs consists of about 8500 PL/I statements (ref. 8).) Schmidt tested EULE using two standard ODE tomes. A comparable test has not been done for ODE2.

Other methods for solving ODEs using MACSYMA have been or are being implemented. Richard Bogen wrote a routine in the MACSYMA language for solving ODEs and systems of ODEs using Laplace transforms. Its top-level routine is called `DESOLVE` and it is described in the file `SHARE;DESOLN USAGE`. It may be loaded into MACSYMA by `LOADFILE(DESOLN,LISP,DSK,SHARE);`. `DESOLVE` may be used for initial value problems as well, and it can handle some equations of greater than second order.

Edward Lafferty is working on a package written in the MACSYMA language for solving ODEs in terms of power series. This work is described in reference 10. (Indeed, Ben Kuipers, the primary author of ODE2, began a series solver as well for Fateman's course.)

One project that yet remains (and which is urged often by Dave Stoutemyer) is to merge these three ODE solvers, using general analytical techniques, Laplace transforms, and series methods, respectively, into one versatile ODE solver so that the user can get the power of all three approaches in one routine.

I wish to thank Ellen Lewis for her helpful assistance.

## APPENDIX

The MACSYMA code for ODE2 follows. (This code comes from the file JPG;ODER 27. Certain less important sections have been omitted.)

```
/*  The Ordinary Differential Equation Solver.
    This package consists primarily of a set of routines taken from Moses'
    thesis and Boyce & DiPrima for solving O.D.E.s of 1st and 2nd order.
    The top-level routines are ODE2, IC1, IC2, and BC2.  */

ODE2(EQ,YOLD,X):=SUBST(YOLD,YNEW,ODE2A(SUBST(YNEW,YOLD,EQ),YNEW,X))$

ODE2A(EQ,Y,X):=BLOCK([DE,A1,A2,A3,A4,Q],
   INTFACTOR: FALSE, METHOD: 'NONE,
   IF FREEOF('DIFF(Y,X,2),EQ)
     THEN IF FTEST(ODE1(EQ,Y,X)) THEN RETURN(Q) ELSE RETURN(FALSE),
   IF DERIVDEGREE(DE: EXPAND(LHS(EQ)-RHS(EQ)),Y,X) # 2
     THEN RETURN(FAILURE(MES1,EQ)),
   A1: COEFF(DE,'DIFF(Y,X,2)),
   A2: COEFF(DE,'DIFF(Y,X)),
   A3: COEFF(DE,Y),
   A4: DE - A1*'DIFF(Y,X,2) - A2*'DIFF(Y,X) - A3*Y,
   IF PR2(A1) AND PR2(A2) AND PR2(A3) AND PR2(A4) AND
      FTEST(HOM2(A1,A2,A3,Y,X))
     THEN IF A4=0 THEN RETURN(Q) ELSE RETURN(VARP(Q,-A4/A1,Y,X)),
   IF FTEST(REDUCE(EQ,Y,X)) THEN RETURN(Q) ELSE RETURN(FALSE))$

ODE1(EQ,Y,X):=BLOCK([DE,F,G,Q],
   IF DERIVDEGREE(DE: EXPAND(LHS(EQ)-RHS(EQ)),Y,X) # 1
     THEN RETURN(FAILURE(MES1,EQ)),
   IF LINEAR2(DE,'DIFF(Y,X)) = FALSE THEN RETURN(FAILURE(MES2,EQ)),
   DE: SOLVE1(DE,'DIFF(Y,X)),
   IF FTEST(SOLVELNR(DE,Y,X)) THEN RETURN(Q),
   IF FTEST(INTFACTOR(G,F,Y,X)) THEN RETURN(EXACT(Q*G,Q*F,Y,X)),
                                 /* LINEAR2 binds F and G */
   IF FTEST(SOLVEHOM(DE,Y,X)) THEN RETURN(Q),
   IF FTEST(SOLVEBERNOULLI(DE,Y,X)) THEN RETURN(Q),
   IF FTEST(GENHOM(DE,Y,X)) THEN RETURN(Q) ELSE RETURN(FALSE))$

PR2(F):=FREEOF(Y,'DIFF(Y,X),'DIFF(Y,X,2),F)$

FTEST(CALL):=IS(NOT((Q: CALL)=FALSE))$

SOLVE1(EQ,Y):=
   BLOCK([DISPFLAG,EQ1],DISPFLAG:FALSE,EQ1:SOLVE(EQ,Y),FIRST(EV(EQ1)))$
```

```
SOLVE2(EQ,Y):=BLOCK([DISPFLAG,EQ1],
   DISPFLAG:FALSE,EQ1:SOLVE(EQ,Y),
   IF NOT(LENGTH(EQ1)=1) THEN RETURN(FAILURE(MES4,EV(EQ1))),
   FIRST(EV(EQ1)))$

MATCHDECLARE([F,G],FREEOF(X))$
DEFMATCH(LINEAR2,F*X+G,X)$

/*  B&D1P, pp. 13-14  */

SOLVELNR(EQ,Y,X):=BLOCK([F,G,W],
   IF LINEAR2(RHS(EQ),Y) = FALSE THEN RETURN(FALSE),
   W: %E^(INTEGRATE(F,X)),
   METHOD: 'LINEAR,
   RETURN(Y=W*(INTEGRATE(G/W,X)+'C)))$

/*  B&D1P, pp. 34-41  */

INTFACTOR(M,N,Y,X):=BLOCK([B1,B2,DMDX,DMDY,DNDX,DNDY,DD],
   DMDX: RATSIMP(DIFF(M,X)),  DMDY: RATSIMP(DIFF(M,Y)),
   DNDX: RATSIMP(DIFF(N,X)),  DNDY: RATSIMP(DIFF(N,Y)),
   IF (DD: DMDY-DNDX) = 0 THEN RETURN(1),
   IF DMDX-DNDY=0 AND DMDY+DNDX=0 THEN RETURN(1/(M^2 + N^2)),
   IF FREEOF(Y, (B1: RATSIMP(DD/N))) THEN RETURN(%E^(INTEGRATE(B1,X))),
   IF FREEOF(X, (B2: RATSIMP(DD/M)))
     THEN RETURN(%E^(INTEGRATE(-B2,Y))) ELSE RETURN(FALSE))$

EXACT(M,N,Y,X):=BLOCK([A,B],
   INTFACTOR: SUBST(YOLD,YNEW,Q),
   A: INTEGRATE(RATSIMP(M),X),
   B: RATSIMP(A + INTEGRATE(RATSIMP(N-DIFF(A,Y)),Y)),
   METHOD: 'EXACT,
   RETURN(B='C))$

/*  B&D1P, pp. 43-44  */

SOLVEHOM(EQ,Y,X):=BLOCK([QQ,A1,A2,A3],
   A1: RATSIMP(SUBST(X*QQ,Y,RHS(EQ))),
   IF NOT(FREEOF(X,A1)) THEN RETURN(FALSE),
   A2: INTEGRATE(1/(A1-QQ),QQ),
   A3: SUBST(Y/X,QQ,A2),
   METHOD: 'HOMOGENEOUS,
   RETURN(RATSIMP('C*X = %E^A3)))$

/*  B&D1P, p. 21, problem 15  */
```

```
SOLVEBERNOULLI(EQ,Y,X):=BLOCK([A1,A2,N],
   A1: COEFF(EQ: EXPAND(RHS(EQ)),Y,1),
   N: HIPOW(RATSIMP(EQ-A1*Y),Y),
   A2: COEFF(EQ,Y,N),
   IF NOT(NUMBERP(N)) OR N=0 OR NOT(EQ = A1*Y + A2*Y^N) THEN RETURN(FALSE),
   A1: INTEGRATE(A1,X),
   METHOD: 'BERNOULLI, ODEINDEX: N,
   RETURN(Y = %E^A1 * ((1-N)*INTEGRATE(A2*%E^((N-1)*A1),X) + 'C) ^ (1/(1-N))))$

/*  Generalized homogeneous equation:  y' = y/x * H(yx^n)
        Reference:  Moses' thesis.  */

GENHOM(EQ,Y,X):=BLOCK([G,U,N,A1,A2,A3],
   G: RHS(EQ)*X/Y,
   N: RATSIMP(X*DIFF(G,X)/(Y*DIFF(G,Y))),
   IF NOT(FREEOF(X,Y,N)) THEN RETURN(FALSE),
   A1: RATSIMP(SUBST(U/X^N,Y,G)),
   A2: INTEGRATE(1/(U*(N+A1)),U),
   A3: RATSIMP(SUBST(Y*X^N,U,A2)),
   METHOD: 'GENHOM, ODEINDEX: N,
   RETURN(X = 'C*%E^A3))$

/*  Chain of solution methods for second order linear homogeneous equations  */

HOM2(A1,A2,A3,Y,X):=
   IF FTEST(CC2(A2/A1,A3/A1,Y,X)) THEN Q ELSE
   IF FTEST(EXACT2(A1,A2,A3,Y,X)) THEN Q ELSE
   IF FTEST(XCC2(A1,A2,A3,Y,X)) THEN Q ELSE FALSE$

/*  B&D1P, pp. 106-112  */

CC2(F,G,Y,X):=BLOCK([A,SIGN,RADPRODEXPAND,ALPHA],
   IF NOT(FREEOF(X,Y,F) AND FREEOF(X,Y,G)) THEN RETURN(FALSE),
   METHOD: 'CONSTCOEFF, RADPRODEXPAND: FALSE,
   SIGN: ASKSIGN(A: F^2-4*G),
   IF SIGN = ZERO THEN RETURN(Y = %E^(-F*X/2) * ('K1 + 'K2*X)),
   IF SIGN = POS THEN
     RETURN(Y = 'K1*%E^((-F+SQRT(A))*X/2) + 'K2*%E^((-F-SQRT(A))*X/2)),
   A: -A, ALPHA: X*SQRT(A)/2,
   IF EXPONENTIALIZE = FALSE THEN
     RETURN(Y = %E^(-F*X/2) * ('K1*SIN(ALPHA) + 'K2*COS(ALPHA))),
   RETURN(Y = %E^(-F*X/2) * ('K1*EXP(%I*ALPHA) + 'K2*EXP(-%I*ALPHA))))$

/*  B&D1P, pp. 98-99, problem 17  */
```

```
EXACT2(A1,A2,A3,Y,X):=BLOCK([B1],
   IF DIFF(A1,X,2) - DIFF(A2,X) + A3 = 0
     THEN B1: %E^(-INTEGRATE((A2 - DIFF(A1,X))/A1, X))
     ELSE RETURN(FALSE),
   METHOD: 'EXACT,
   RETURN(Y = 'K1*B1*INTEGRATE(1/(A1*B1),X) + 'K2*B1))$

/*  B&DiP, pp. 113-114, problem 16  */

XCC2(A1,A2,A3,Y,X):=BLOCK([D,B1],
   IF A3=0 THEN RETURN(FALSE),
   D: RATSIMP((A1*DIFF(A3/A1,X) + 2*A2*A3/A1)/(2*(A3/A1)^(3/2))),
   IF FREEOF(X,Y,D) THEN B1: CC2(D,1,Y,Z) ELSE RETURN(FALSE),
   METHOD: 'XFORMTOCONSTCOEFF,
   RETURN(SUBST(INTEGRATE(SQRT(A3/A1),X),Z,B1)))$

/*  B&DiP, pp. 124-127  */

VARP(SOLN,G,Y,X):=BLOCK([Y1,Y2,Y3,Y4,WR],
   Y1: RATSIMP(SUBST(['K1=1,'K2=0],RHS(SOLN))),
   Y2: RATSIMP(SUBST(['K1=0,'K2=1],RHS(SOLN))),
   WR: Y1*DIFF(Y2,X) - Y2*DIFF(Y1,X),
   IF WR=0 THEN RETURN(FALSE),
   Y3: RATSIMP(Y1*G/WR),
   Y4: RATSIMP(Y2*G/WR),
   YP: RATSIMP(Y2*INTEGRATE(Y3,X) - Y1*INTEGRATE(Y4,X)),
   METHOD: 'VARIATIONOFPARAMETERS,
   RETURN(Y = RHS(SOLN) + YP))$

/*  Methods to reduce second-order equations free of x or y  */

REDUCE(EQ,Y,X):=BLOCK([B1,QQ],
   B1: SUBST(['DIFF(Y,X)=QQ, 'DIFF(Y,X,2)=QQ], EQ),
   IF FREEOF(Y,B1) THEN RETURN(NL1(EQ,Y,X)),
   IF FREEOF(X,B1) THEN RETURN(NL2(EQ,Y,X)) ELSE RETURN(FALSE))$

/*  B&DiP, p. 89, problem 1  */

NL1(EQ,Y,X):=BLOCK([DE,B,A1,A2,V],
   DE: SUBST(['DIFF(Y,X)=V, 'DIFF(Y,X,2)='DIFF(V,X)], EQ),
   IF (B: ODE1(DE,V,X)) = FALSE THEN RETURN(FALSE),
   A1: SUBST([V='DIFF(Y,X),'C='K1], B),
   A2: SOLVE2(A1,'DIFF(Y,X)),
   IF A2=FALSE THEN RETURN(FALSE),
   IF FTEST(ODE1(A2,Y,X))
```

```
     THEN (METHOD: 'FREEOFY, RETURN(SUBST('K2,'C,Q))) ELSE RETURN(FALSE))$

/*  B&D1P, p. 89, problem 2  */

NL2(EQ,Y,X):=BLOCK([DE,B,A1,A2,YZ,V],
   DE: SUBST(['DIFF(Y,X)=V, 'DIFF(Y,X,2)=V*'DIFF(V,YZ), Y=YZ], EQ),
   IF (B: ODE1(DE,V,YZ)) = FALSE THEN RETURN(FALSE),
   A1: SUBST([V='DIFF(Y,X),YZ=Y,'C='K1], B),
   A2: SOLVE2(A1,'DIFF(Y,X)),
   IF A2=FALSE THEN RETURN(FALSE),
   IF FTEST(ODE1(A2,Y,X))
     THEN (METHOD: 'FREEOFX, RETURN(SUBST('K2,'C,Q))) ELSE RETURN(FALSE))$

IC1(SOLN,XC,YC):=
   EV(SOLN, C=RHS(SOLVE1(EV(SOLN,XC,YC),C)), RATSIMP)$

BC2(SOLN,XA,YA,XB,YB):=BLOCK([DISPFLAG,SINGSOLVE,TEMP],
   DISPFLAG:FALSE, SINGSOLVE:TRUE,
   TEMP: MAP(LAMBDA([ZZ], EV(SOLN,ZZ,EVAL)),
             SOLVE([EV(SOLN,XA,YA), EV(SOLN,XB,YB)], ['K1,'K2])),
   IF LENGTH(TEMP)=1 THEN RETURN(FIRST(TEMP)) ELSE RETURN(TEMP))$

IC2(SOLN,XA,YA,DYA):=BLOCK([DISPFLAG,SINGSOLVE,TEMP],
   DISPFLAG:FALSE, SINGSOLVE:TRUE,
   TEMP: LHS(SOLN) - RHS(SOLN),
   TEMP: MAP(LAMBDA([ZZ], EV(SOLN,ZZ,EVAL)),
             SOLVE([EV(SOLN,XA,YA), SUBST([DYA,XA],
                   LHS(DYA)=-SUBST(0,LHS(DYA),DIFF(TEMP,LHS(XA)))
                               /DIFF(TEMP,LHS(YA)))],
                 ['K1,'K2])),
   IF LENGTH(TEMP)=1 THEN RETURN(FIRST(TEMP)) ELSE RETURN(TEMP))$

FAILURE(MES,EQ):=(LDISP(SUBST(YOLD,YNEW,EQ)), DISP(MES), FALSE)$

MES1:    "NOT A PROPER DIFFERENTIAL EQUATION"$
MES2:    "FIRST ORDER EQUATION NOT LINEAR IN Y'"$
MES3:    "CANNOT DETERMINE SIGN OF CONSTANT EXPRESSION"$
MES4:    "MULTIPLE SOLUTIONS TO FIRST PARTIAL PROBLEM"$
```

## REFERENCES

1. Moses, J.: Symbolic Integration. Ph.D. Thesis, Massachusetts Inst. Technol., Dec. 1967. (Also available as Report MAC TR-47.)

2. Boyce, W. E.; and DiPrima, R. C.: Elementary Differential Equations. Second ed., John Wiley & Sons, 1969.

3. Moses, J.: Symbolic Integration: The Stormy Decade. Commun. ACM, vol. 14, no. 8, Aug. 1971, pp. 548-560.

4. The Mathlab Group: MACSYMA Reference Manual. Version 8. Lab. Comput. Sci., Massachusetts Inst. Technol., Nov. 1975.

5. Fateman, R.: An Approach to Automatic Asymptotic Expansions. Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation, Aug. 1976, pp. 365-371.

6. Stoutemyer, D.: Computer Algebraic Manipulation for the Calculus of Variations, the Maximum Principle, and Automatic Control. ALOHA System Technical Report A74-5, Univ. of Hawaii, Nov. 1974.

7. Bogen, R.: A Program for the Solution of Integral Equations. Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper no. 2 of this compilation.)

8. Schmidt, P.: Automatic Symbolic Solution of Differential Equations of First Order and First Degree. Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation, Aug. 1976, pp. 118-125.

9. Schmidt, P.: Maschinelle Symbolische Losung von Differentialgleichungen I Ordnung und I Grades. Doktorgrades dissertation, Univ. Bonn, 1976.

10. Lafferty, E.L.: Power Series Solution of Ordinary Differential Equations. Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper no. 34 of this compilation.)

# A PROGRAM FOR THE SOLUTION OF INTEGRAL EQUATIONS*

Richard A. Bogen
University of Hawaii

## SUMMARY

This paper is intended to supplement an article by Stoutemyer (ref. 1) which describes a program for the solution of one dimensional integral equations. The program, first written in REDUCE (ref. 2) has been implemented in MACSYMA with several additional techniques which are explained herein. By utilizing many methods, the program can obtain closed-form and series solutions to a large class of linear and non-linear problems. One of the techniques developed, reduction to a differential equation, has not previously appeared in the literature in the general form described in this paper.

## INTRODUCTION

The interface between a person and a computer system may be considered to take place on many possible "levels", as distinguished by the facilities most often used. In a symbolic mathematics system, e.g. MACSYMA, there are system designers who work mainly in LISP offering an initial set of MACSYMA functions. The application programmers in turn use these functions to construct others which are useful to the casual users who need to solve some particular problem by invoking a sequence of built-in functions. The ease with which each level of user can accomplish his task is dependent on how complete and well-designed the facilities are at all lower levels. Ideally there should be no need for a user at one level to program at a lower level. The arguments for using a pre-written program rather than writing one from scratch are as strong as those for using a computer in the first place as opposed to hand calculation; i.e. it saves time and affords less chance of making an error.

Some of the most useful programs are those for solving certain types of equations. MACSYMA already contains routines for solving various kinds of algebraic and ordinary differential equations. In reference 1, Stoutemyer describes a program he wrote in the REDUCE symbolic mathematics language (ref. 2) for solving integral equations. In order to make this facility available to users of MACSYMA, an implementation was begun in July 1976.

---

*This work was supported by the National Science Foundation under Grant No. MCS75 - 22983.

Since completing this, we have discovered several new techniques and these have been added to the program. This paper is intended to supplement the work reported in reference 1, so the emphasis here is on the new techniques that are not described there. Following a discussion of these techniques an outline of the program is given, some limitations are mentioned, and a comparison is made with the earlier REDUCE version. Finally some planned future improvements are described. A demonstration is presented in the appendix.

## TECHNIQUES

The types of integral equations considered by the program are those reducible to the "quasi second-kind":

$$p(x) = v\left(x, p(x), \int_{a(x)}^{b(x)} w(x,u,p(x),p(u))\, du\right)$$

and the "first-kind":

$$\int_{a(x)}^{b(x)} w(x,u,p(u))\, du = f(x)$$

where, for this paper, p(x) is the unknown function, x is the independent variable, and u is the integration variable.

The original REDUCE program contains five techniques applicable to certain second-kind equations, two for certain first-kind equations, and two usable for both types of equations. These are summarized in table I. Since implementing these in MACSYMA, a further search of the literature turned up two additional first-kind techniques.

Kanwal (ref. 3) gives a generalization of Abel's method for singular integrands of the form:

$$\int_a^x \frac{p(u)\, du}{(h(x) - h(u))^k} = f(x), \qquad 0 < k < 1$$

The solution is:

$$p(x) = \frac{\sin(k\pi)}{\pi} \frac{d}{dx} \int_a^x \frac{h'(u) f(u)\, du}{(h(x) - h(u))^{1-k}} \qquad \text{where } h' \text{ denotes } \frac{dh}{du}.$$

Cochran (ref. 4) gives a method for linear fixed-limit first-kind equations with finite-rank integrands. These have the form:

$$\int_a^b \left[\sum_{j=1}^{n} q_j(x) r_j(u)\right] p(u)\, du = f(x).$$

Letting the coefficient of $q_j(x)$ in $f(x)$ be $c_j$ and assuming $p(u)$ to be:

$$\sum_{k=1}^{n} d_k r_k(u)$$

the problem is reduced to that of solving the n simultaneous linear equations:

$$c_j = \sum_{k=1}^{n} d_k \int_a^b r_j(u) r_k(u)\, du \qquad j=1,2,\ldots,n$$

for the $d_k$. This gives one solution. The result of adding to this linear combinations of functions orthogonal to all of the $r_k(u)$ gives additional solutions.

In addition to the two techniques mentioned thus far, one other has been made available. Stoutemyer proposed a generalization of a method in Goursat (ref. 5) for transforming any variable-limit finite-rank integral equation into an ordinary differential equation. It is applicable to both first-kind and second-kind equations. There are numerous methods for solving differential equations and MACSYMA already possesses routines implementing some of these methods. Consequently, this reduction significantly enlarges the class of integral equations for which exact solutions can be obtained. The method is remarkably simple. We are given an integral equation of the form:

$$\left\{\begin{array}{c} f(x) \quad \text{or} \\ p(x) - f(x) \end{array}\right\} = \int_a^x \sum_{j=1}^{n} q_j(x) r_j(u, p(u))\, du.$$

Letting $R_j(x) = \int_a^x r_j(u, p(u))\, du$, we have:

$$\left\{\begin{array}{c} f(x) \quad \text{or} \\ p(x) - f(x) \end{array}\right\} = \sum_{j=1}^{n} q_j(x) R_j(x). \tag{1}$$

Equation (1) together with its first n-1 derivatives with respect to x gives a set of n simultaneous equations linear in the n unknowns $R_j(x)$, j=1,2,...,n. Solving these equations and substituting for the $R_j(x)$ in the $n^{th}$ derivative of equation (1) gives an ordinary differential equation for p(x) which is of order n-1 or n depending on whether the left side of equation (1) was f(x), for first-kind, or p(x) - f(x), for second-kind. Initial conditions can be obtained by setting x=a in equation (1) and its derivatives, then solving successively for p(a), p'(a), ... ,$p^{m}(a)$, where m is n-2 or n-1 as above.

We illustrate this technique with a non-trivial example. Consider:

$$-\frac{9x^6}{20}+\frac{5x^4}{6}-\frac{x^2}{4}-\frac{2x}{15}=\int_1^x (ux^2+u^2x)p(u)\,du.$$

Letting $R_1(x)=\int_1^x u\,p(u)\,du$ and $R_2(x)=\int_1^x u^2p(u)\,du$ yields:

$$-\frac{9x^6}{20}+\frac{5x^4}{6}-\frac{x^2}{4}-\frac{2x}{15}=x^2R_1(x)+xR_2(x) \tag{2}$$

Taking two successive derivatives gives:

$$-\frac{27x^5}{10}+\frac{10x^3}{3}-\frac{x}{2}-\frac{2}{15}=2x^3p(x)+2xR_1(x)+R_2(x) \tag{3}$$

$$-\frac{27x^4}{2}+10x^2-\frac{1}{2}=2x^3p'(x)+9x^2p(x)+2R_1(x). \tag{4}$$

Solving equations (2) and (3) for $R_1(x)$ we have:

$$R_1(x)=\frac{-9x^4+10x^2-1}{4}-2x^2p(x).$$

Substituting this into equation (4) and re-arranging terms results in:

$$2x\,p'(x)+5\,p(x)=-9\,x^2+5$$

whose solution is $p(x)=cx^{-5/2}+1-x^2$. To solve for c we let x=1 in equation (3) and, noting that $R_1(1)=R_2(1)=0$, find that $p(1)=0$, which implies c=0.

## EXAMINATION OF THE PROGRAM

The program is invoked by the calling sequence:

IEQN(expression, unknown, technique, napprox, guess).

The first argument is the integral equation. Trailing arguments may be omitted, in which case they will assume default values which are:

unknown - defaults to the first function encountered in an integrand which is unknown to MACSYMA.

technique - defaults to FIRST which causes all applicable techniques to be tried until one succeeds (see below).

napprox - defaults to 1 and represents the maximum number of iterations or adjustable collocation parameters for an approximate solution.

guess - defaults to NONE and represents the initial guess for NEUMANN or FIRSTKINDSERIES techniques. If NONE, the initial guess will be the value obtained by setting all integrals in the expression to zero.

The method used by the program is to factor the first argument to IEQN and for each factor containing an integral the equation "factor = 0" is algebraically solved for the unknown in terms of the other parts of the factor. If a solution results, then "second-kind" techniques are tried. Otherwise the program tries "first-kind" techniques. These techniques are listed below in outline form giving conditions under which they are applicable. (The name of the technique, which can be used as the third argument of IEQN, is capitalized.)

## Second-Kind Techniques

(Exact)

- Constant limits of integration (Fredholm type)
    - Finite-rank integrand - FINITERANK
- A constant lower limit and x as the upper limit (Volterra type)
    - Integrand linear in p(u)
        - Convolution integral - TRANSFORM (Laplace transform)
    - Finite-rank integrand - DIFFEQN (Conversion to ODE)

(Approximate)

- Arbitrary limits of integration
    - Integrand linear in p(u) - FREDSERIES
    - There exists a point at which the limits are equal - TAYLOR
    - NEUMANN
    - COLLOCATE

## First-Kind Techniques

(Exact)

- Constant limits of integration
    - Integrand linear in p(u)
        - Finite-rank integrands - FINITERANK
- A constant lower limit and x as the upper limit
    - Integrand linear in p(u)
        - Abel's equation - ABEL
        - Convolution integral - TRANSFORM
    - Finite-rank integrand - DIFFEQN

(Approximate)

- Arbitrary limits of integration
    - FIRSTKINDSERIES
    - COLLOCATE

It is difficult to make an accurate comparison between the execution times of the MACSYMA and REDUCE versions for several reasons. The PDP-10 processor on which MACSYMA runs is significantly faster and has more memory space resulting in fewer garbage collections. Also the REDUCE versions of the SOLVE, INTEGRATE, and LAPLACE routines were interpreted rather than compiled and REDUCE includes display generation times in its figures. Consequently, the execution times for the examples given in reference 1 were around 10 times the figures obtained when these examples were run on MACSYMA.

The text of the program was approximately 30% smaller on MACSYMA due to the availability of more built-in functions. Naturally, the MACSYMA version could handle more cases because of more comprehensive integration, equation-solving, and transform routines.

At present, the major difficulty in using the integral equation solver is the frequent exhaustion of available storage due to the loading of files containing many auxiliary functions which are not part of the initial system. Indeed, a single problem may cause functions in a dozen such files to be referenced. Once loaded, the space they occupy cannot be re-used even if they are no longer needed. In this situation, the user can save relevant values, load a fresh MACSYMA, and continue where he left off. If, however, all the space was consumed in a single call to IEQN, because of attempting several solution techniques, then the user should try separate calls for each one. It is unlikely that this approach will cause difficulty since the principal limitations of particular techniques arise not from space or time constraints, but from the inability of some functions to handle certain kinds of arguments. In particular, for linear integral equations the trouble spots are the inverse Laplace transform, which is limited to rational functions, and the ordinary differential equation solver which is limited to first and second order equations. Thus the corresponding cases of convolution equations containing non-polynomial functions and of finite-rank integrals with rank greater than two can only be handled by the approximate methods. For non-linear finite-rank equations, solutions can be found only if corresponding non-linear differential equations or algebraic equations can be solved.

## FUTURE IMPROVEMENTS

Aside from alleviating the problems mentioned in the previous section, there are a number of ways in which the program could be extended. Eigen-analysis as well as testing existence and uniqueness theorems could automatically provide useful information even when no solution can be determined. Integral transforms such as those of Fourier and Mellin and the Wiener-Hopf technique would enable the program to be used for some important integrals with infinite limits. Finally, the program could be made to handle systems of integral equations thus greatly extending its applicability. Incorporation of these techniques is under current investigation.

## REFERENCES

1. Stoutemyer, D.: Analytical Solution of Integral Equations Using Computer Algebra. Trans. Math. Software, June 1977.

2. Hearn, A.C.: REDUCE 2 Users's Manual. Computational Physics Group, Univ. Utah, 1973.

3. Kanwal, R.P.: Linear Integral Equations: Theory and Technique. Academic Press, 1971.

4. Cochran, J.A.: The Analysis of Linear Integral Equations. McGraw-Hill Book Co., 1972.

5. Goursat, E.: Integral Equations - Calculus of Variations, A Course in Mathematical Analysis. Vol. III, Pt. 2, Dover Publ. Inc, 1964.

```
(C1)  'INTEGRATE(P(U)/(X**2-U**2)**(1/3),U,0,X)=X;

                           X
                          /
                          (      P(U)
(D1)                      I  ------------ DU = X
                          )     2   2 1/3
                          /  (X  - U )
                           0

(C2)  IEQN(D1)$
DEFAULT 2ND ARG, THE UNKNOWN: P(X)
DEFAULT 3RD ARG, TECHNIQUE: FIRST
DEFAULT 4TH ARG, NUMBER OF ITERATIONS OR COLLOCATION PARAMETERS: 1
DEFAULT 5TH ARG, INITIAL GUESS FOR NEUMANN OR FIRSTKINDSERIES: NONE

                                 1   2/3
                   SQRT(3) GAMMA(-) X
                                 3
(E2)              [---------------------, ABEL ]
                                     5
                   2 SQRT(%PI) GAMMA(-)
                                     6

(C3)  P(X)=1-'INTEGRATE(P(U)**2,U,1,X)$

(C4)  IEQN(D3,P(X),DIFFEQN,1,NONE)$

                          1
(E4)           [ P(X) = -----, DIFFEQN, [ X = 1, P(X) = 1 ] ]
                        X + C

(C5)  P(X)=2*X+'INTEGRATE(X*U*P(U),U,0,1)$

(C6)  IEQN(D5,P(X),ALL,2,NONE)$

(E8)                              [ 3 X, FINITERANK ]

(E9)                            [ 3 X, FREDSERIES, 2 ]
                           26 X
(E10)                     [ ----, NEUMANN, 2, APPROXIMATE ]
                            9

(E11)                           [ 3 X, COLLOCATE, 2 ]
```

TABLE I - SUMMARY OF TECHNIQUES PREVIOUSLY REPORTED ON (in ref. 1)

| Name | Form to which applicable | Method |
|---|---|---|
| FINITERANK | 2nd-kind, fixed limits, finiterank integrands. | Given $p(x)$="expr", distribute integration in expr over all sums, then replace each integral of $q_j(x)r_j(u,p(u))$ by $c_jq_j(x)$ where $c_j$ is an arbitrary parm. to be determined. This gives $p(x)=g(x)$. Then solve the n simul. lin. eqns.<br>$c_j=\int_a^b r_j(u,g(u))\,du$<br>for the $c_j$, $j=1,\ldots,n$. |
| " | 1st or 2nd-kind, rank-1, variable limits. | Special cases of the DIFFEQN method for a rank-1 integral. |
| TRANSFORM | 1st or 2nd-kind, convolution, variable limits. | Take Laplace trans., solve for trans. of $p(x)$, then invert. |
| FREDSERIES | 2nd-kind, linear. | Given $p(x)=f(x)+\int K(x,u)p(u)du$, the solution is $p(x)=f(x)+\int G(x,u)f(u)du$, where $G(x,u)$ is the quotient of two infinite series whose terms are found from recurrence relations. |
| TAYLOR | 2nd-kind, variable limit. | Given $p(x)=f(x)+\int_{a(x)}^{b(x)} w(x,u,p(u))du$ find a point c where $a(c)=b(c)=c$. Expand $p(x)-f(x)$ in Taylor series about $x=c$ by differentiation. |
| NEUMANN | 2nd-kind. | Make a guess for $p(x)$ and iterate using original equation. |
| FIRSTKINDSERIES | 1st-kind. | |
| COLLOCATE | any. | Assume a particular form for $p(x)$ involving n arbitrary parameters. Substitute in equation and evaluate at n values of x to get a set of simul. eqns. to solve for parms. |
| Differentiation | 1st-kind, var. limit. | Differentiate given equation some number of times to see if a 2nd-kind equation results. |

BLANK PAGE

BLANK PAGE

# SYMBOLIC LAPLACE TRANSFORMS OF SPECIAL FUNCTIONS *

Yannis Avgoustis
Laboratory for Computer Science (formerly Project MAC)
Massachusetts Institute of Technology

## ABSTRACT

A MACSYMA implementation of the Laplace Transform for Special Functions is described. The Generalized Hypergeometric Functions are used as a basis for the representation of approximately fifty Special Functions. Only a relatively small number of formulas that generally involve Generalized Hypergeometric Functions are utilized for the integration stage.

A sample of actual examples and their timing is provided at the end of the paper.

## I INTRODUCTION

We describe a design for the Laplace Transform of Special Functions which has been implemented in MACSYMA (ref. 1). In our design we have employed approximately all of the fifty well known Special Functions, known also as the Functions of Mathematical Physics (ref. 2), (ref. 3). In designing the Laplace Transform capability, we have considered it as part of the "definite integration" problem and our design is planned to cover a significant part of definite integration through interaction at some later time with the other Integral Transforms, such as Hankel, Y, K, Fourier, Mellin, etc .

One faces two main difficulties when dealing with this problem. First, definite integration generally is a recursively unsolvable problem (ref. 4). Second, the area of Special Functions is well known for its "chaotic state" (ref. 5).

Wang and Bogen have also worked on the problem of definite integration (ref. 6) and Laplace Transforms (ref. 7). However, they both were interested

---

* This work was supported, in part, by ERDA contract Number E(11-1)-3070 and NASA Grant NSG 1323.

mainly in Elementary Functions. To the best of our knowledge there has been no other system designed for any of the integral transforms or definite integration for the Special Functions.

In our design we take advantage of the fact that most of the Special Functions can be considered as particular instances of the Generalized Hypergeometric Function and therefore can be integrated, using the Generalized Hypergeometric Function representation, with a table consisting of very few formulas. A natural consequence is that the result of the integration procedure involves Generalized Hypergeometric Functions. Hence an additional step is required to reduce the Generalized Hypergeometric Functions into Special or/and Elementary Functions.

## II THE GENERAL IDEA

We begin with the definitions of the Generalized Hypergeometric Functions (ref. 8), (ref. 2), and the Laplace Transforms (ref. 9), (ref. 10).

Definition 1. We call the Generalized Hypergeometric Function, otherwise known as the Generalized Gauss function, the series

$$ {}_pF_q[a_1,a_2,\ldots,a_p;b_1,b_2,\ldots,b_q;z] = \sum_{n=0}^{\mathrm{inf}} \frac{(a_1)_n(a_2)_n \ldots (a_p)_n}{(b_1)_n(b_2)_n \ldots (b_q)_n} \frac{z^n}{n!} \tag{1} $$

where $a_1$, $a_2,\ldots,$ $a_p$ and $b_1$, $b_2,\ldots,b_q$ are complex parameters, z is a complex variable. We also denote the above series as ${}_pF_q[a_1,a_2,\ldots,a_p;b_1,b_2,\ldots,b_q;z]$ or ${}_pF_q[(a);(b);z]$ or simply ${}_pF_q(z)$.

The series ${}_pF_q(z)$ satisfies the differential equation

$$ \Big(z \frac{d}{dz}\Big(z \frac{d}{dz} + b_1-1\Big)\Big(z \frac{d}{dz} + b_2 - 1\Big) \ldots \Big(z \frac{d}{dz} + b_q - 1\Big) \tag{2} $$

$$ - z\Big(z \frac{d}{dz} + a_1\Big)\Big(z \frac{d}{dz} + a_2\Big) \ldots \Big(z \frac{d}{dz} + a_p\Big)\Big) y = 0 $$

Definition 2. We call the Laplace Transform of a real or complex function f(t), defined for all real nonnegative values of t, the integral

$$ \int_0^\infty f(t)e^{-pt}\, dt \tag{3} $$

if it exists for some values of the complex variable p. It is written L[f(t)] and determines a function F(p); thus

$$L[f(t)] = \int_0^\infty f(t)e^{-pt}\,dt = F(p) \tag{4}$$

The key ideas in our design, depicted in figure 1, are

Stage 1. Represent the Special Functions, if possible, as particular instances of the Generalized Hypergeometric Function.

Stage 2. Provide a fairly general formula to integrate the results of stage 1.

Stage 3. Take the result of stage 2 involving a Generalized Hypergeometric Function, and reduce it to an elementary or/and Special Function(s).

Hence, our design alternates between two levels:

Level 1. The expression involves Special or/and Elementary Functions.

Level 2. The expression involves Generalized Hypergeometric Functions.

We next proceed with a simple illustration of the above scheme.

Illustration

Given input

$$t^{-3/2}\, I_3(2a^{1/2}t^{1/2})\, e^{-pt} \tag{5}$$

where $I_3(.)$ is a modified Bessel function of the first kind (ref. 11), (ref. 12), the following will take place in each of the three stages:

Stage 1.

Because

$$I_v(z) = e^{-v\pi i/2}\, J_v(ze^{\pi i/2}) \tag{6}$$

(5) becomes

$$i t^{-3/2}\, J_3(2ia^{1/2}t^{1/2})\, e^{-pt} \tag{7}$$

Because

$$J_v(z) = \frac{z^v}{2^v\Gamma(v+1)}\, {}_0F_1[\ v+1;\ -1/4\ z^2] \tag{8}$$

(7) becomes

$$\frac{a^{3/2}}{6p}\ {}_0F_1[\ 4;\ at]\ e^{-pt} \tag{9}$$

Stage 2.

In this stage we integrate by using the following formula (ref. 13)

$$\int_0^\infty t^{s-1}{}_mF_n(a_1,\dots,a_m;r_1,\dots,r_n;(lt)^k)e^{-pt}\,dt \tag{10}$$

$$= \Gamma(s)p^{-s}{}_{m+k}F_n(a_1,\dots,a_m,\frac{s}{k},\frac{s+1}{k},\dots,\frac{s+k-1}{k};r_1,r_2,\dots,r_n;(\frac{kl}{k})^k)$$

which is valid for $Re(s) > 0$, $m+k < n+1$, where $k,m,n$ are integers.

Thus (9) becomes

$$\frac{a^{3/2}}{6p}\ {}_1F_1[\ 1;\ 4;\ a/p] \tag{11}$$

Stage 3.

At stage 3, we apply to (11) the following "Kummer's transformation" (ref. 2)

$${}_1F_1[\ a;\ r;\ z] = e^z\ {}_1F_1[\ r-a;\ r;\ -z] \tag{12}$$

and (11) reduces to

$$\frac{a^{3/2}}{6p}\ e^{a/p}\ {}_1F_1[\ 3;\ 4;\ -a/p] \tag{13}$$

We recognize that the series in (13) is an instance of an Incomplete Gamma function (ref. 2), because

$${}_1F_1[\ a;\ a+1;\ -x] = ax^{-a}\ \gamma(a,x) \tag{14}$$

Therefore, (14) finally becomes

$$\frac{e^{a/p}\ p^2}{2a^{3/2}}\ \gamma(3,a/p) \tag{15}$$

## III THE GENERALIZED HYPERGEOMETRIC FUNCTION AND THE FUNCTIONS OF MATHEMATICAL PHYSICS

As we have already mentioned, we have dealt with around fifty Special Functions and our goal is to interpret them as particular instances of the Generalized Hypergeometric Function.

We have divided the set of the Special Functions into two major types. The first type includes all Special Functions that are directly transformed through some relation into a Generalized Hypergeometric Function, and the second type includes those functions that are expressed in terms of other Special Functions and ultimately are expressed in terms of Special Functions of the first type. This is the major objective of the first stage and it has been influenced by the tendency to utilize and manipulate as few Special Functions as is necessary.

For example, the Bessel function of the first kind $J_v(z)$ belongs to the first type and is automatically transformed into a Generalized Hypergeometric Function through relation (8).

The Hankel function of the first kind, $H_{v,1}(z)$, is expressed initially as a sum of a first and second kind of Bessel functions as it is shown in (16)

$$H_{v,1}(z) = J_v(z)+iY_v(z) \tag{16}$$

Here $J_v(z)$ is a function of the first type, while $Y_v(z)$, a Bessel function of the second kind, is not. $Y_v(z)$ is transformed in terms of $J_v(z)$ as long as v is not an integer through the relation

$$Y_v(z) = (\cos(v\ pi)J_v(z) - J_{-v}(z))\ \csc(v\ pi) \tag{17}$$

Thus we have ultimately expressed $H_{v,1}(z)$ in terms of the first type function $J_v(z)$, which in turn can readily be transformed into a Generalized Hypergeometric Function. The case in which v is an integer, $Y_v(z)$ is considered separately.

In a similar way we have considered products of Special Functions which can be expressed as a single Generalized Hypergeometric Function. Thus the product of two Bessel functions "$J_v(z)*J_m(z)$" is a product belonging to the first type and is transformed into a Generalized Hypergeometric Function through the relations (8) and (18)

$${}_0F_1[\; r;\; z]\,{}_0F_1[\; s;\; z] = {}_2F_3[\; r/2+s/2,\; r/2+s/2-1/2;\; r,\; s,\; r+s-1;\; 4z] \tag{18}$$

On the other hand, the product $I_v(z)*K_m(z)$ ,where $I_v(z)$, $K_m(z)$ are modified Bessel functions of the first and second kind, respectively, belongs to the second type and is ultimately expressible in terms of functions of the first type, for noninteger values of the index m.

## IV LAPLACE TRANSFORMS

A design for the Laplace Transform algorithm should incorporate two major components: the integration process, and the different Laplace Transforms properties.

We decided to form a table which contains as few formulas as possible. This strategy has the following consequences:

1. The overall design of the system becomes algorithmic in the sense that the system works deterministically and knows what it can really do and what it cannot, and does not waste time by trying different approaches.

2. The main burden and difficulty of the problem shifts from stage 2 to stage 1 and especially stage 3, where we have to reduce the Generalized Hypergeometric Functions to some Elementary or/and Special Function(s).

As far as the Laplace Transforms properties are concerned, our general policy consists of applying them in stage 2, in the Generalized Hypergeometric Function level. Hence, stage 2 can be divided into two substages.

Substage 2.1 Utilize the Laplace Transforms properties.

Substage 2.2 Integrate.

This policy changes only in cases where such a postponment of the application of the Laplace Transforms properties until stage 2, causes irreparable damage and failure in our scheme (figure 1). Therefore the Laplace Transforms properties have been considered in two types. Properties of the first type can be applied in substage 2.1, independently of what kind of Special Function(s) that the input expression contains. Thus, for example, all the well known properties, such as the "scale property" (ref. 10)

$$L[e^{-at}f(t)] = F(p+a) \tag{19}$$

belong to the first type.

Properties of the second type cannot be applied after stage 1 for certain Special Functions and our scheme is unable to proceed successfully to stages two and three. For example, the property

$$L[f(a\sinh t)] = \int_0^{\infty} J_p(au)g(u)\,du \tag{20}$$

where g(p) = L[f(t)], cannot be applied after stage 1, for the Bessel function $J_0$, as in, for example,

$$J_0(a\sinh t)\, e^{-pt} \qquad (21)$$

since after the completion of the first stage we get

$$_0F_1[\,1;\ -\frac{a^2}{4}\sinh^2 t]e^{-pt} \qquad (22)$$

Expression (22) cannot be integrated since our table does not contain any formulas with such functional arguments while it is too late to apply property (20).

The above mentioned example could be solved by two recursive calls to our scheme (figure 1). First, by calling the scheme as described for the Laplace Transforms, and second by calling the same scheme in which the Laplace Transforms properties and Integration formulas have been substituted with Hankel Transforms properties and Integration formulas (ref. 9).

On a first examination, a program that can take the Laplace Transforms of approximately fifty Special Functions would imply that quite a big number of formulas would be necessary to be incorporated in the table look-up of our second stage. It turns out that relatively very few formulas are needed. Thus, formula (10) has been applicable to a large number of Special Functions (ref. 14), (ref. 2), (ref. 3), namely the Bessel Functions of the first and second kind, both Modified Bessel Functions, the two kinds of Hankel Functions, also the Struve functions, the Lommel functions, and the Kelvin functions, the Whittaker, the error and both Incomplete Gamma functions, for almost all the values of their indices and for linear and quadratic functions of the argument. Furthermore, in cooperation with general formulas of other Integral Transforms, formula (10) contributes in integrating composite functions like $J_0(\sinh t)$, as we have already shown.

Currently, our table look-up incorporates seven formulas and our design is generally capable of integrating expressions described in the two categories below:

1. Special Functions of linear or quadratic argument multiplied with

    a. Arbitrary powers of the argument.

    b. Trigonometric and exponential functions of linear argument.

2. Products of two Special Functions of linear or quadratic argument, multiplied with the same kind of functions we mentioned in the first category. The Special Functions of this category can be functions of only one of the following groups:

a. Any kind of Bessel, Modified Bessel, or Hankel functions.

b. Orthogonal Polynomials.

c. Confluent Hypergeometric Functions.

However, the potentiality of keeping very few formulas around in the table of our second stage would be of limited value if we were unable to complete successfully the third stage, to reduce the Generalized Hypergeometric Function to some Elementary or/and Special Function(s).

## V THE REDUCTION STAGE.

In the reduction stage the Generalized Hypergeometric Function is reduced, if that is possible, to some Elementary or/and Special Function(s). Priority is always given first to those methods that reduce the Series into Elementary Functions and then to those that reduce to the most common Special Functions, such as error, Bessel etc . The effort in the reduction stage increases as the number of the series parameters, and subsequently the p and q values, increase. If the reduction is unsuccessful then the series ${}_pF_q(z)$ is returned.

The reduction stage incorporates two phases. In the first phase algorithms independent of the values of p and q of the series ${}_pF_q(z)$ are applied. In the second phase special algorithms dependent on the parameters are performed.

A surprisingly useful rule, incorporated in the first reduction phase, is the following.

If a numerator parameter of the series ${}_pF_q(z)$ exceeds by a positive integer, say k, a denominator parameter, then the series ${}_pF_q(z)$ can be expressed as the sum of k+1 ${}_{p-1}F_{q-1}(z)$ 's.

Such a series splitting, though it does not actually fully reduce a ${}_pF_q(z)$, simplifies the reduction by decreasing the p and q values.

To illustrate series splitting, consider

$$t^3 J_0(t^{1/2})^2 e^{-pt} \tag{23}$$

after stages one and two have been completed, we get

$$6p^{-4} {}_3F_3[ 1/2, 1, 4; 1, 1, 1; p^{- 1}] \tag{24}$$

Now, at stage three and after a trivial general reduction rule, (24) becomes

$$6p^{-4} {}_2F_2[ 1/2, 4; 1, 1; -p^{-1}] \tag{25}$$

then applying our general "splitting" rule, (25) reduces to

$$6p^{-4} [ {}_1F_1[ 1/2; 1; -p^{-1}] - 3/2 p^{-1} {}_1F_1[ 3/2; 2; -p^{-1}] \tag{26}$$
$$+ 9/16 p^{-2} {}_1F_1[ 5/2; 3; -p^{-1}] - 5/96 p^{-3} {}_1F_1[ 7/2; 4; -p^{-1}]]$$

which ultimately yields

$$6p^{-4} e^{-1/2 p^{-1}} [ I_0(-1/2 p^{-1}) + 3/2 M_{-1/2,1/2}(-p^{-1}) \tag{27}$$
$$+ 9/16 p^{-2} (-p)^{3/2} M_{-1,1}(-p^{-1}) - 5/96 p^{-1} M_{-3/2,3/2}(-p^{-1})]$$

where $M_{i,j}$ is a Whittaker function.

In the second phase, reductions are easy for the cases ${}_0F_0(z)$, ${}_0F_1(z)$, ${}_1F_0(z)$, and the difficulty increases significantly for higher p's and q's. We have been mainly concerned with the Confluent Hypergeometric Function reduction, ${}_1F_1(z)$, and the Gauss Hypergeometric Functions, ${}_2F_1(z)$, that include, in addition to certain important Special Functions, the Elementary Functions. The most important tools here are the different transformations: linear, quadratic, etc (ref. 15), (ref. 2), (ref. 16), and the Contiguous Functions Relations (ref. 17).

The different transformations (linear, quadratic, etc) are performed as soon as it is detected that the Generalized Hypergeomtric Function is reducible to some other ones and which are definitely known to be reducible to some Special or Elementary Functions in one or more steps. We clarify the above ideas in a simple example, where a quadratic transformation is applied to a Gauss Hypergeometric Function.

Suppose we are given

$$ {}_2F_1[\mathrm{alpha},\mathrm{beta};\ \mathrm{gamma};\ \mathrm{arg}] = {}_2F_1[\ 3/4,\ 5/4;\ 1/2;\ z^2] \qquad (28) $$

where

$$ \mathrm{beta} - \mathrm{alpha} = 5/4 - 3/4 = 1/2 \qquad (29) $$

therefore the quadratic transformation

$$ \left(1 - \frac{z}{2}\right)^{-a} {}_2F_1\left[\frac{a}{2}, \frac{a}{2} + \frac{1}{2};\ b + \frac{1}{2};\ \frac{z^2}{(2-z)^2}\right] = {}_2F_1[\ a,\ b;\ 2b;\ z] \qquad (30) $$

is applicable. Hence, the following relation holds:

$$ {}_2F_1\left[\ 3/4,\ 5/4;\ 1/2;\ \frac{z^2}{(2-z)^2}\right] = \left(1 - \frac{z}{2}\right)^{3/2} {}_2F_1[\ 3/2,\ 0;\ 0;\ z] \qquad (31) $$

Upon application of a simple general reduction rule, the right hand side of expression (31) becomes

$$ (1 - z/2)^{3/2}\ {}_1F_0[\ 3/2;\ ;\ z] \qquad (32) $$

and finally, taking into account the relation

$$ {}_1F_0[\ a;\ ;\ z] = (1 - z)^{-a} \qquad (33) $$

expression (28) reduces to

$$ \frac{1}{6(1+z)^{3/2}} + \frac{5}{6(1-z)^{3/2}} \qquad (34) $$

"Contiguity" has been also found useful and has been put into use in the reduction of the Generalized Hypergeometric Functions.

Definition. We call two Generalized Hypergeometric Functions contiguous if they are alike except for one pair of parameters in which they differ by a unity.

Thus the Hypergeometric Function ${}_2F_1[a,b;c;z]$ is contiguous to ${}_2F_1[a+1,b;c;z]$ and obviously to only five others. Any three of the contiguous functions can be connected with a linear relation, the so called Contiguous Functions (Recurrence) Relations. Such relations are applied to a Generalized Hypergeometric Function whenever it has been predetermined that the resulting series can be reduced to Special or/and Elementary Functions.

Given

$$ {}_1F_1[\,-1/2;\ 3/2;\ z] \qquad (35)$$

and using the following contiguous relation

$$(a-c+1)\ {}_1F_1[a;\ c;\ z] - a\ {}_1F_1[a+1;\ c;\ z] + (c-1)\ {}_1F_1[a;\ c-1;\ z] = 0 \qquad (36)$$

we get

$$1/2\ {}_1F_1[\,1/2;\ 3/2;\ z] + 1/2\ {}_1F_1[\,-1/2;\ 1/2;\ z] \qquad (37)$$

where the first series is identified as an error and the second as an Incomplete Gamma function, namely

$$-1/4\ iz\pi^{1/2}\ \ \mathrm{Erf}(iz^{1/2}) - 1/2\ z^{1/2}\ \gamma(-1/2,-z) \qquad (38)$$

Similarly, the Hypergeometric Function

$${}_2F_1[\,a,\ a+9/2;\ c;\ z] \qquad (39)$$

can be reduced through successive use of the contiguous relations to the following sum

$$\begin{aligned}
&\frac{(c-a-9/2)(c-a-7/2)(c-a-5/2)(c-a-3/2)}{(c-2a-9/2)(c-2a-7/2)(c-2a-5/2)(c-2a-3/2)}\ {}_2F_1[\,a,\ a+1/2;\ c;\ z] \qquad (40)\\
&- 4\ \frac{a(c-a-9/2)(c-a-7/2)(c-a-5/2)(1-z)}{(c-2a-11/2)(c-2a-9/2)(c-2a-7/2)(c-2a-3/2)}\ {}_2F_1[\,a+1,\ a+3/2;\ c;\ z]\\
&+ 6\ \frac{a(a+1)(c-a-9/2)(c-a-7/2)(1-z)^2}{(c-2a-13/2)(c-2a-11/2)(c-2a-7/2)(c-2a-5/2)}\ {}_2F_1[\,a+2,\ a+5/2;\ c;\ z]\\
&- 4\ \frac{a(a+1)(a+2)(c-a-9/2)(1-z)^3}{(c-2a-15/2)(c-2a-11/2)(c-2a-9/2)(c-2a-7/2)}\ {}_2F_1[\,a+3,\ a+7/2;\ c;\ z]\\
&+ \frac{a(a+1)(a+2)(a+3)(1-z)^4}{(c-2a-15/2)(c-2a-13/2)(c-2a-11/2)(c-2a-9/2)}\ {}_2F_1[\,a+4,\ a+9/2;\ c;\ z]
\end{aligned}$$

We next notice that the parameters of each of the above Hypergeometric Series satisfy a similar relation to (29). Therefore a quadratic transformation is applicable to each of them, that ultimately leads to the following sum of Legendre functions

$$2^{c-1}\ \Gamma(c)\ z^{(1-c)/2}\ (1-z)^{(c-2a-1)/2} \tag{41}$$

$$\left[\frac{(c-a-9/2)(c-a-7/2)(c-a-5/2)(c-a-3/2)}{(c-2a-9/2)(c-2a-7/2)(c-2a-5/2)(c-2a-3/2)} P_{c-2a-1,1-c}((1-z)^{-1/2})\right.$$

$$- 4\ \frac{a(c-a-9/2)(c-a-7/2)(c-a-5/2)}{(c-2a-11/2)(c-2a-9/2)(c-2a-7/2)(c-2a-3/2)} P_{c-2a-3,1-c}((1-z)^{-1/2})$$

$$+ 6\ \frac{a(a+1)(c-a-9/2)(c-a-7/2)}{(c-2a-13/2)(c-2a-11/2)(c-2a-7/2)(c-2a-5/2)} P_{c-2a-5,1-c}((1-z)^{-1/2})$$

$$- 4\ \frac{a(a+1)(a+2)(c-a-9/2)}{(c-2a-15/2)(c-2a-11/2)(c-2a-9/2)(c-2a-7/2)} P_{c-2a-7,1-c}((1-z)^{-1/2})$$

$$\left. + \frac{a(a+1)(a+2)(a+3)}{(c-2a-15/2)(c-2a-13/2)(c-2a-11/2)(c-2a-9/2)} P_{c-2a-9,1-c}((1-z)^{-1/2})\ \right]$$

## VI COMMENTS AND CONCLUSIONS.

The Laplace Transforms package is relatively fast, as the actual examples in the appendix show. Furthermore, it is capable of quickly rejecting cases that it cannot process.

The Laplace Transforms system is capable of providing results for the well known Special Functions limited to essentially linear and quadratic arguments. However, cases like equation (21), mentioned earlier, or the following one

$$t^{-1}\ J_1(at^{-1})\ e^{-pt} \tag{42}$$

are some of those that the present Laplace Transforms implementation is unable to provide an answer, unless it will interact properly with other Integral Transforms. We expect to generalize the system to those other transforms in the coming year.

Currently our system is able to solve approximatelly 80% of the entries of the corresponding chapters of the Tables of Integral Transforms (The Bateman's Manuscript Project). We expect to be able to cover 3/4 of the remaining cases in the coming months by increasing the capabilities of our first and third stages. Finally, we should add in favor of our implementation, its capability to integrate expressions that are only implicitly included in Bateman's Manuscript Project.

# APPENDIX

This is a sample of some actual examples of the Laplace Transform system in MACSYMA. "Definte" is the top function that calls the integral transforms, it takes two arguments: the expression to be integrated and the variable, and assumes limits of integration from zero to infinity.

```
/* Laplace transforms */

ASSUME(P > 0);
(D11)                             [P > 0]

(C12) TIME:TRUE$
TIME= 1 MSEC.

(C13) /* Some "Confluents".
"M[k,m](z)" is a Whittaker function.
"GAMMAINCOMPLETE(a,b)", and "GAMMAGREEK(a,b)" are current names
for the Incomplete Gamma functions: Γ(a,b), and γ(a,b). */

%E^(A*T)*T^2*ERF(T^(1/2))*%E^(-P*T);
TIME= 22 MSEC.
                                          2   A T - P T
(D13)                       ERF(SQRT(T)) T  %E

(C14) DEFINTE(%,T);

RPART FASL DSK MACSYM being loaded
load      done
Is      P  positive, negative, or zero?

NEGATIVE;

GAMMA FASL DSK MAXOUT being loaded
loading done
TIME= 431 MSEC.
(D14)
               1                       2                           1
   15 (--------------- - ----------------------- + --------------------------)
               1                       1      3/2              2    1       5/2
       SQRT(----- + 1)   7 (P - A) (----- + 1)     21 (P - A)  (----- + 1)
            P - A                   P - A                       P - A
  ---------------------------------------------------------------------------
                                                 7/2
                                        4 (P - A)

(C15) T^(1/2)*GAMMAINCOMPLETE(1/2,A*T)*%E^(-P*T);
TIME= 632 MSEC.
                                    1                 - P T
(D15)                GAMMAINCOMPLETE(-, A T) SQRT(T) %E
                                    2
```

```
(C16) DEFINTE(%,T);
TIME= 1586 MSEC.
                       %PI                         2
(D16)      --------------------------- - ------------------------
                3/2        A   3/2             3/2       A   3/2
           2 (P + A)   (1 - -----)     (P + A)     (1 - -----)
                            P + A                       P + A

(C19) T^(3/2)*M[1/2,1](T)*%E^(-P*T);
TIME= 12 MSEC.
                                      3/2   - P T
(D19)                      M       (T) T     %E
                            1/2, 1

(C20) DEFINTE(%,T);
TIME= 263 MSEC.
                           1                1
                    6 (--------- + ----------------------)
                             1            1          1  2
                       1 - -----   4 (P + -) (1 - -----)
                               1          2          1
                           P + -                 P + -
                               2                     2
(D20)               ------------------------------------
                                         1 4
                                    (P + -)
                                         2

(C21) /* Some Bessel functs (bf's). */
/* J[v](z), 1st kind of bf's. */
/* Y[v](z), 2nd kind of bf's.*/
/* H[v,1](z), 1st kind of the 3rd kind of bf's (1st Hankel). */
/* H[v,2](z), 2nd kind of the 3rd kind of bf's (2nd Hankel).*/

T^(-1/2)*J[0](2*A^(1/2)*T^(1/2))*%E^(-P*T);
TIME= 16 MSEC.
                                                 - P T
                        J (2 SQRT(A) SQRT(T)) %E
                         0
(D21)                   ---------------------------
                                  SQRT(T)

(C22) DEFINTE(%,T);
TIME= 256 MSEC.
                                                A
                                              - ---
                                       A        2 P
                          SQRT(%PI) I (---) %E
                                     0 2 P
(D22)                     -----------------------
                                  SQRT(P)

(C23) T^(3/2)*Y[1](A*T)*%E^(-T);
TIME= 9 MSEC.
                                      3/2   - T
(D23)                         Y (A T) T    %E
                               1
```

```
(C24) DEFINTE(%,T);
TIME= 968 MSEC.
                                         %I     1       3/4
                15 %I SQRT(2) P         (- --) (------ - 1)
                               - 2, 1/2   A      2
                                                A  + 1
(D24)           -------------------------------------------
                                2     2    2     2      1/4
                  8 SQRT(%PI) (A  + 1)  ((A  + 1)  - 1)

(C25) T^(3/2)*H[1/2,1](T)*%E^(-P*T);
TIME= 12 MSEC.
                                       3/2   - P T
(D25)                      H      (T) T    %E
                            1/2, 1

(C26) DEFINTE(%,T);
TIME= 1389 MSEC.
                                                     1          2
                                      %I SQRT(2) (------ - -------------)
                                                  1          1      2 2
                                                  -- + 1   3 (-- + 1)  P
                                                   2          2
                    4                             P          P
(D26) ----------------------------- - -----------------------------------
                         1     2  3                           2
      SQRT(2) SQRT(%PI) (-- + 1)  P               SQRT(%PI) P
                          2
                         P


(C29) /* I[v](z), K[v](z), Modified bf's. */

T^(1/2)*I[1](T)*%E^(-P*T);
TIME= 11 MSEC.
                                            - P T
(D29)                       I (T) SQRT(T) %E
                             1

(C30) DEFINTE(%,T);
TIME= 295 MSEC.
                                       1            1       5/2
           3 SQRT(%PI) P          (-----------) SQRT(-- - 1) P
                        - 3/2, - 1         1            4
                                  SQRT(1 - --)         P
                                            2
                                           P
(D30)      -----------------------------------------------------
                                    16

(C31) T^(5/2)*K[1/2](T)*%E^(-P*T);
TIME= 12 MSEC.
                                          5/2   - P T
(D31)                        K    (T) T    %E
                              1/2

(C32) DEFINTE(%,T);
TIME= 1761 MSEC.
```

```
                                                                  4              1
        3 (%I - 1) (%I + 1) SQRT(2) SQRT(%PI) (-------------- + ---------)
                                                      1  3  2         1  2
                                                5 (1 - --)  P     (1 - --)
                                                        2              2
                                                       P              P
(D32)   ------------------------------------------------------------------
                                       4
                                    2 P

                                                                4              1
                (%I - 1) (%I + 1) SQRT(2) SQRT(%PI) (-------------- + ---------)
                                                          1  3  2         1  2
                                                    3 (1 - --)  P     (1 - --)
                                                            2              2
                                                           P              P
             - ------------------------------------------------------------------
                                                   3
                                                2 P


(C35) I[0](2*A^(1/2)*T^(1/2))^2*%E^(-P*T);
TIME= 15 MSEC.
                                                      2   - P T
(D35)                      I  (2 SQRT(A) SQRT(T))  %E
                            0

(C36) DEFINTE(%,T);
TIME= 944 MSEC.
                                          2 A
                                          ---
                                    2 A    P
                                 I  (---) %E
                                  0  P
(D36)                            -------------
                                       P

(C37) J[1/2](T^(1/2))*Y[1/2](T^(1/2))*%E^(-P*T);
TIME= 15 MSEC.
                                                           - P T
(D37)                    J    (SQRT(T)) Y    (SQRT(T)) %E
                          1/2             1/2

(C38) DEFINTE(%,T);
TIME= 366 MSEC.
                                                1
                                              - ---
                                      1         2 P
                             %I I    (---) %E
                                 1/2 2 P
(D38)                      - --------------------
                                      P

(C39) I[1/2](T^(1/2))*K[1/2](T^(1/2))*%E^(-P*T);
TIME= 15 MSEC.
                                                           - P T
(D39)                    I    (SQRT(T)) K    (SQRT(T)) %E
                          1/2             1/2

(C40) DEFINTE(%,T);
```

```
TIME= 2938 MSEC.
                                   1                             1
                                  ---                           ---
                            1     2 P                     1     2 P
      %I %PI (%I + 1) I   (---) %E       %PI (%I + 1) I  (---) %E
                       1/2 2 P                      1/2 2 P
(D40) -------------------------------- - ---------------------------
                    4 P                            4 P

                                             1                              1
                                            ---                            ---
                                      1     2 P                      1     2 P
                %I %PI (%I - 1) I   (---) %E     %PI (%I - 1) I   (---) %E
                                 1/2 2 P                       1/2 2 P
               + ------------------------------ + ---------------------------
                               4 P                          4 P

(C41) /* Related to bf's functions. */
/* Struve functions. */

T^(-1/2)*LSTRUVE[-1/2](T^(1/2))*%E^(-P*T);
TIME= 16 MSEC.
                                               - P T
                        LSTRUVE     (SQRT(T)) %E
                               - 1/2
(D41)                   -----------------------------
                                   SQRT(T)

(C42) DEFINTE(%,T);
TIME= 1196 MSEC.
                                                                  1
                                                                 ---
                                          1       3       1      8 P
           (%I - 1) (%I + 1) SQRT(2) GAMMA(-) GAMMA(-) I   (---) %E
                                          4       4   1/4 8 P
(D42)    - ----------------------------------------------------------
                              4 SQRT(%PI) SQRT(P)

(C43) T*HSTRUVE[1](T)*%E^(-P*T);
TIME= 10 MSEC.
                                            - P T
(D43)                       HSTRUVE (T) T %E
                                   1

(C44) DEFINTE(%,T);
TIME= 229 MSEC.
                                    16 %I
(D44)                     - ----------------------
                                 3/2  1      3/2  3
                            3 %PI    (-- + 1)    P
                                       2
                                      P

(C45) /* Lommel functions. */

T^(1/4)*S[1/2,-1/2](T^(1/2))*%E^(-P*T);
TIME= 15 MSEC.
                                           1/4   - P T
(D45)                 S          (SQRT(T)) T    %E
                       1/2, - 1/2
```

```
(C46) DEFINTE(%,T);
TIME= 226 MSEC.
                                                         1
                                                      - ---
                                                        4 P
                     %I SQRT(%PI) ERF(- 2 %I SQRT(P)) %E
(D46)              - --------------------------------------
                                       3/2
                                     2 P
```

## REFERENCES

1. The Mathlab Group: MACSYMA Reference Manual. Lab. Comput. Sci., Massachusetts Inst. Technol., Nov. 1975.

2. Erdelyi, Magnus; and Oberhettinger, Tricomi: Higher Transcendental Functions. Bateman Manuscript Project - Volumes 1,2 and 3, McGraw-Hill Book Co., 1953.

3. Hochstadt, H.: The Functions of Mathematical Physics. Interscience Publ., 1971.

4. Wang, P. S.: The Undecidability of the Existence of Zeros of Real Elementary Functions. J. Assoc. Comput. Mach., vol. 21, no 4, Oct. 1974, pp. 586-589.

5. Vilenkin, N. J.: Special Functions and the Theory of Group Representations. Translations of Mathematical Monographs, vol. 22, American Math. Soc., 1968.

6. Wang, P. S.: Evaluation of Definite Integrals by Symbolic Manipulation. Ph.D Thesis, Massachusetts Inst. Technol., Oct. 1971. (Also available as TR-92)

7. Bogen, R. A.: Automatic Computation of direct and inverse Laplace Transforms using Computer Symbolic Mathematics. Proceedings of the 10th Hawaii International Conference on Systems Sciences, Jan. 1977, pp. 161-169.

8. Slater, L.: Generalized Hypergeometric Functions. Cambridge Univ. Press, 1966.

9. Sneddon, I. H.: The Use of Integral Transforms. McGraw-Hill Book Co., 1972.

10. Hladik, J: La Transformation de Laplace a plusiers variables. Masson et Cie, 1969.

11. Watson, G. N.: A Treatise on the Theory of Bessel Functions. Cambridge Univ. Press. 1952.

12. Tranter, C. J.: Bessel Functions. Hart Pub. Co., Inc., 1969.

13. Erdelyi, Magnus; and Oberhettinger, Tricomi: Tables of Integral Transforms. Bateman Manuscript Project - Volumes 1 and 2, McGraw-Hill Book Co., 1954.

14. Buchholz, H.: The Confluent Hypergeometric Function. Springer Tracts Nat. Philos., vol. 15, 1969.

15. Abramowitz, M.; and Stegun, Irene A.: Handbook of Mathematical Functions, Dover Publ. Inc., 1965.

16. Goursat, E. M.: Sur l' equation differentielle lineaire qui admet pour integrale la serie hypergeometrique. Ann. Sci. Ecole Norm. Sup. (2), 10, 3-142. 1881.

17. Rainville, E. D.: The Contiguous Function Relations for ${}_pF_q$. Bull. American Math. Soc., vol. 51, 1945, pp. 714-723.

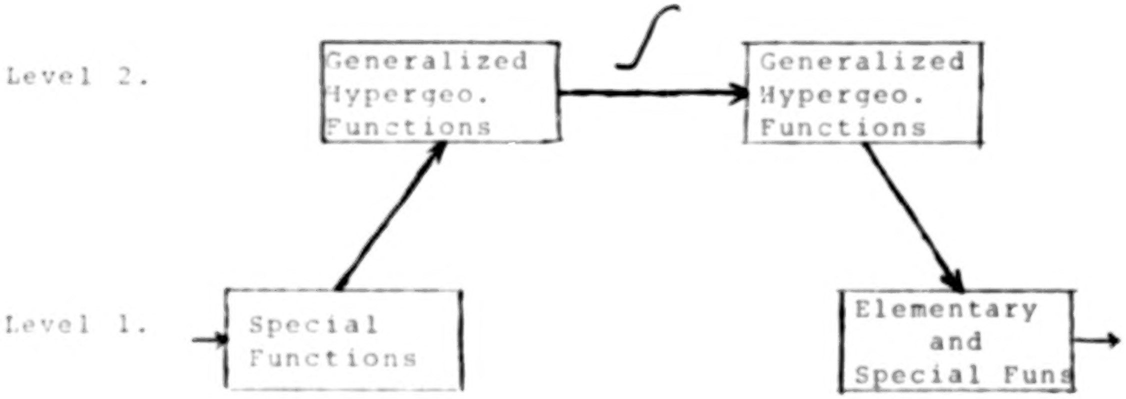

Level 2.
Generalized Hypergeo. Functions
∫
Generalized Hypergeo. Functions
Level 1.
Special Functions
Elementary and Special Funs

Figure 1.

BLANK PAGE

BLANK PAGE

# AN IMPROVED ALGORITHM FOR THE ISOLATION OF POLYNOMIAL REAL ZEROS*

Richard J. Fateman

University of California
Berkeley, California 94720

## SUMMARY

The Collins-Loos algorithm for computing isolating intervals for the zeros of an integer polynomial requires the evaluation of polynomials at rational points. This implies the use of arbitrary precision integer arithmetic. This paper shows how careful use of single-precision floating-point arithmetic within the context of a slightly modified algorithm can make the calculation considerably faster and no less exact. Typically, 95% or more of the evaluations can be done without exact arithmetic. The precise speed-up depends on the relative costs of the arithmetic in a given implementation. Our implementation on the DEC KL-10 computer is some 5 to 10 times faster than the original Univac 1110 implementation in SAC-I. We are able to attribute about a factor of three improvement to the MACSYMA machine and language, and 2.7-3.3 speed-up to the algorithm itself.

## 1. INTRODUCTION

Collins and Loos (reference 1) sketch an algorithm, and provide some implementation details for computing a set of intervals on the real line (a1,b1],...,(an,bn] such that each interval contains a single or multiple real zero of a polynomial P. The multiplicity of the ith interval is also computed. This algorithm requires the exact evaluation of P and its derivatives at rational points. For most of the algorithm, one is actually unconcerned about the value of P or its derivatives, since the sign (+1, 0, or -1) is sufficient to determine whether P is above, on, or below the x-axis.

The sign may be determined, as shown in section 2, by a procedure using primarily floating-point arithmetic; in case the sign cannot be so determined, either higher precision or exact rational arithmetic is used. It might be tempting to dismiss this technique as being "machine dependent", and so it is; however, the dependency is isolated to a single floating-point value representing the maximum relative error in the result of a floating point operation. We know of no computer for which this number cannot be determined.

---

* The work described herein was performed with the help of MACSYMA which is supported, in part, by the United States Energy Research and Development Administration under Contract Number E(11-1)-3070 and by the National Aeronautics and Space Administration under Grant NSG 1323.

In places where exact values are computed in ref. 1, we are usually able, through the use of (pessimistic) floating-point interval arithmetic (ref. 2) to avoid the attendant cost of exact arithmetic. In fact, most of the reliance on exact arithmetic demonstrated in the tests (duplicating those in (ref. 1)) is generated by exponent overflow rather than insufficient accuracy.

## 2. HORNER'S RULE WITH ERROR BOUNDS

Assume we wish to evaluate a polynomial $p(z) = \sum_{j=0}^{n} a_j z^{n-j}$ at a point $z = x$.

$$p(z) = p(x) + (z-x) \sum_{j=0}^{n-1} b_j z^{n-1-j} \qquad (2.1)$$

where Horner's recurrence provides the $b_j$'s:

$$b_0 = a_0$$

$$b_j = x\, b_{j-1} + a_j,\ j = 1,2,\ldots,n-1$$

$$\text{and } b_n = p(x)$$

Assume we are using arithmetic subject to truncation and round-off error.

Then for some small constants $\alpha_j$, $\beta_j$, the computed value of $b_j$ is

$$b_j = (x\, b_{j-1} (1 + \beta_{j-1}) + a_j)/(1 + \alpha_j) \qquad (2.2)$$

(assume $b_{-1} \equiv 0$ for the following)

$$p(z) = \sum_{j=0}^{n} a_j z^{n-j} = \sum_{j=0}^{n} \left[(1 + \alpha_j)\, b_j - x\, b_{j-1} (1 + \beta_{j-1})\right] z^{n-j}$$

$$= \sum_{j=0}^{n} (1 + \alpha_j)\, b_j z^{n-j} - x \sum_{j=0}^{n-1} b_j (1 + \beta_j) z^{n-1-j}$$

$$= b_n (1 + \alpha_n) z^0 + \sum_{j=0}^{n-1} b_j \{(1 + \alpha_j) z - (1 + \beta_j) x\} z^{n-1-j} \qquad (2.3)$$

Application of (2.1) provides, at $z = x$

$$b_n(1 + \alpha_n) = p(x) - \sum_{j=0}^{n-1} b_j\ (\alpha_j - \beta_j)\ x^{n-j} \qquad (2.4)$$

Thus the magnitude of the error $|b_n - p(x)| = \left|\sum_{j=0}^{n-1} b_j(\alpha_j - \beta_j)\ x^{n-j} + b_n\alpha_n\right|$.

Since $|\alpha_j|$, $|\beta_j| \leq \epsilon$, $\epsilon$ a unit in the last place, ( $\epsilon = 2^{-27}$ on a 27-bit base 2 mantissa machine such as the PDP-10),

the bulk of the error is $\leq 2\ \epsilon \sum_{j=0}^{n} |b_j|\ \ |x|^{n-j}$ (2.5)

The above analysis, due to W. Kahan, can be extended to complex values of x (ref. 2 and 3).

We wish to extend the analysis to include approximation of x by a floating point representation, and approximation of each $a_j$ by a floating point representation.

That is $x = \hat{x}\ (1 + \delta)$, $a_j = \hat{a}_j\ (1 + \gamma_j)$.

An alternative to (2.2) is then

$$b_j = (x(1 + \delta)b_{j-1}\ (1 + \beta_{j-1}) + a_j(1 + \gamma_j)/(1 + \alpha_j) \qquad (2.2')$$

which becomes, analogous to (2.3):

$$p(z) = b_n\left(\frac{1+\alpha_n}{1+\gamma_n}\right) + \sum_{j=0}^{n-1} b_j\left\{\left(\frac{1+\alpha_j}{1+\gamma_j}\right)z - \frac{(1+\delta)(1+\beta_j)}{(1 + \gamma_{j-1})}\,x\right\}\ z^{n-1-j} \qquad (2.3')$$

Following the analysis to (2.4) yields

$$b_n(1+\alpha_n) = (1+\gamma_n)\left[p(x) - \sum_{j=0}^{n-1} b_j\left\{\left(\frac{\alpha_j-\gamma_j}{1 + \gamma_j}\right) - \frac{(\delta+\beta_j-\gamma_{j-1}+\delta\beta_j)}{1+\gamma_{j-1}}\right\}x^{n-j}\right]$$

Thus the magnitude of the error, neglecting terms which are products of two small terms is bounded by $\hat{e}$, the rhs of the equation below:

$$|b_n - p(x)| \leq$$

$$|\gamma_n\ p(x)| + |b_n\alpha_n| + \left|\sum_{j=0}^{n-1} b_j\ \{\alpha_j-\gamma_j+\gamma_{j-1}-\delta-\beta_j\}x^{n-j}\right| \leq 5\ \epsilon \sum_{j=0}^{n} |b_j|\ \ |x|^{n-j} \qquad (2.5')$$

Typically the integer coefficients of p will be representable exactly as floating point numbers, as will x (since x typically is an exact binary fraction resulting from bisection of intervals with binary fraction end points) so that $\delta$ and the $\gamma_j$ will frequently be zero.

It may be argued that we have calculated $\hat{e}$ imprecisely, but the rhs of (2.5') is a sum of positive terms and the error involved can be shown to be a second order effect. Being pessimistic, we use $6\varepsilon$ rather than $5\varepsilon$ as a coefficient so as to be positive of bounding the error.

Thus if we wish to find the sign of a polynomial p at a point x, we evaluate $\hat{p}(\hat{x})$ and $\hat{e}$, the error bound. If $\hat{e} \geq |\hat{p}(\hat{x})|$, then we do not know the sign definitely. We can re-evaluate to higher precision: how much higher can be estimated from equation (2.5). If $p(x) = 0$, we will have to use rational arithmetic to prove it; thus if $\hat{p}(\hat{x}) = 0$, a direct test for zero using rational arithmetic would be needed.

## 3. IMPLEMENTATION

A first draft of this paper and a MACSYMA implementation were mentioned in a talk at the SYMSAC conference, August, 1976 (ref. 4). Since that time, Professor Loos was kind enough to supply an ALDES language version of the program described in (ref. 1). After correcting a few typographical errors presumably not present in the SAC-I program, it was possible to duplicate the results of (ref. 1) fairly closely. We were not able to achieve exactly the same numbers of evaluations, a situation which we believe arises because the SAC-I program differs in some respects from the ALDES description. This duplication was done by writing in MACSYMA's Algol-60-like language, followed by semi-automatic translation to LISP, followed by compilation to machine language.

Certain programs were already in existence in MACSYMA, and did not have to be programmed for this application; these included some involved with the detection of floating point overflows. In step 4 below, one minor improvement was achieved by a simple 4 line assembly language alteration. This amounted to 1% in total time. All other programming was done in higher level languages such as LISP.

The MACSYMA implementation running on a DEC-KL-10 computer seems to run faster than the SAC-I implementation on the UNIVAC 1110 by a factor of 3 or more; this, using the most faithful recreation of the algorithm as seemed appropriate. Computing a strict isolation list for the 5th Legendre polynomial, L[5] required .74 seconds in SAC-I, .128 seconds in MACSYMA. For L[25] the times were 35 and 11 seconds, respectively. An attempt to divorce these numbers from storage allocation time may make the comparison more relevant: if SAC-I spends 1/3 of its time in such bookkeeping (a figure suggested by Prof. Loos), and MACSYMA spent 5 of the 11 seconds in LISP "garbage collection" (gc) by actual measurement, then the two systems compare at 23 and 6 seconds respectively. We suspect that MACSYMA's host system has relatively faster multiple-precision integer arithmetic, resulting in these shorter times.

Improvements to the Collins-Loos algorithm proceeded in several steps.

Step 1:

All computations of polynomial signs were attempted in single-precision floating-point arithmetic, first. No exact values were computed except when needed (equations 24 and 25 of (ref. 1)), when the error in the floating-point evaluation was too high to determine the sign, or an exponent overflow occurred during the sign computation. Note that some polynomials can never be evaluated without overflow in single-precision because their coefficients are too large to be expressed in the floating point range. For such cases we must use some other technique: exact rational arithmetic, approximate unlimited-exponent arithmetic such as MACSYMA's "bigfloat" system, or some other algorithm entirely. (The DEC-10 floating-point format specifies a 27-bit fraction, 8-bit (excess 128) exponent, and 1-bit sign. Arithmetic is base 2 (not 8 or 16).)

For the same polynomial, L[25], 93.7% of the arithmetic could be done in single-precision floating-point. The time was reduced from 11 seconds to about 7.2 (2.5 in gc). As noted in (ref. 1), these polynomials can be handled very rapidly by a Sturm-sequence base root-finder, and in fact MACSYMA's took 7.5 seconds (4.3 in gc) on this polynomial.

Incidentally, the speed difference between SAC-I and MACSYMA on Sturm-sequences is also about 3:1.

Step 2:

Computations were done in single-precision initially, then in multiple precision when possible, otherwise using exact arithmetic. The software multiple precision (ref. 4) removes the need to check for exponent overflow in Horner's rule, but incurs a higher cost than the binary rational arithmetic advocated by Collins and Loos, in some cases. (In fact, binary rational arithmetic is very similar to floating point arithmetic, the difference being that the "fraction" is of varying length, and is exact. If that length is small, the floating-point arithmetic will be comparatively more expensive. For L[30], the "longest" binary rational endpoint of an isolating interval a/b is only 8 bits long in a and b, suggesting that floating point is at a disadvantage here.)

For L[25] again, 93.7% of the arithmetic could be done in single-precision, another 4.6% in multiple-precision, and only 1.8% in exact arithmetic. Considering the fact that L[25] cannot even be evaluated at its computed root bound (16) without overflow in single-precision, this seems fairly impressive.

Step 3:

It is possible to eliminate all exact computations within the scope of the algorithm by replacing the tangent construction in (ref. 1) by a procedure suggested in the earlier draft of this paper requiring only evaluation with rigorous error bounds. It was hoped that this removal of all exact arithmetic would speed up the computation. The alternative of using essentially the same tangent algorithm but with floating-point interval arithmetic, and when necessary, exact arithmetic was more successful. Although it was possible to reduce the number of exact evaluations to a very small number (e.g., 20 of some 1300 for L[30]), some of the floating point multiple precision evaluations were slower than exact evaluation at a binary-rational point.

It appears that single-precision floating-point interval arithmetic nearly always is sufficient, in the tests suggested by Collins and Loos. Most decisions can be made with this arithmetic and it is faster than multiple-precision. In the table below we do not tabulate the multiple-precision measurements. When an interval calculation is insufficiently precise for a decision, we revert momentarily back to exact evaluation for isolating interval computation for that polynomial derivative. A more elaborate algorithm would use exact evaluation for redoing exactly the smallest computation that failed, but we did not choose this technique, because of algorithm complexity.

Step 4:

Since so much of the computation is done in floating point, we sought to decrease the time spent in arithmetic by open-compiling floating point arithmetic in the one short program implementing Horner's rule.

This is easily done in MACLISP, but at the expense of loss of overflow detection. Four instructions were inserted in the compiler-generated LAP (LISP Assembly Program) code for the Horner's rule program to reset flags at the beginning and at the end test (once) for overflow in any of the operations. The coefficients in the polynomial and its derivatives were also converted to floating point, once in the main loop. In case this could not be done because of overflow, the original version of the algorithm was used for that polynomial derivative under consideration. These changes sped up the run-time considerably, to about 2.3 seconds for L[25] (plus gc). This is 10 times faster than the original program running on the UNIVAC 1110, and 2.2 times faster than our own version of the Collins-Loos algorithm.

## 4. EMPIRICAL TESTS

The tests in table 1 are representative of a larger class of tests with randomly generated polynomials, at least in the relative timing of the various zero-finding programs being compared. Further comparisons, including additional work mentioned in section 5, should be forthcoming.

## 5. ANALYSIS

By comparison with (ref. 1), we may add only a few items of interest.

Since the "worst case" for our algorithm is similar to the Collins-Loos worst case, we can only observe that empirically, most calculations were not "worst case" and could be done in single-precision floating point. The major problem, that of overflow, could be handled by more elaborate scaling procedures, such as carrying an additional word for the exponent. We did not pursue this. The time for finding all real zeros of a polynomial of degree n is likely to be on the average, under our algorithm, $0(n^3)$ by the same arguments as in (ref. 1).

We expect further progress in this area can be made in two directions: Given the isolating intervals, it may be shown that a Newton-iteration's convergence can be assured, using starting points developed from the strict isolation list of the second derivative of the polynomial of interest; also, the vast difference in time for finding these intervals versus numerically approximating the roots is disturbing. Since the library programs for polynomial zero approximation using standard numerical procedures are an order of magnitude faster, it seems reasonable to obtain approximations in this way, and then "prove" the locations of the zeros, and their multiplicities after the fact. Bruce Char, a Berkeley graduate student has worked on this problem using a simple technique described in reference 5. It is not clear that we could compute even the one greatest-common-divisor calculation to remove multiple roots in less time than we could find all isolating intervals for the roots by the numerical methods currently available. Char's program appears to be much faster than the functionally similar program described in reference 6. For example, the roots of the 13th cyclotomic polynomial are isolated by Pinkert's algorithm in 220 seconds on a PDP-10. Char's routine takes less than 0.5 seconds on a PDP-10 (perhaps a model 4 times faster than Pinkert's). Char's routine must sometimes defer to other methods such as described here, where floating point yields to exact calculation, but this is only when its internal checks demonstrate that the isolation of all complex roots (currently, of a real polynomial) has not been achieved.

As the program currently exists, it is faster than Sturm sequence calculations on most polynomials with few real roots, and thus should be used in place of that zero-finder, except when it is known in advance that many real zeros exist. Since the numerical programs are so much faster, we expect that the usefulness of this program is quite restricted, in terms of the typical MACSYMA user, to those applications where misdiagnosis of a zero would have special dire consequences in the course of a computation, and furthermore, the polynomial is known in advance to be numerically difficult.

We are grateful to Prof. W. Kahan for numerous discussions on this topic.

## 6. REFERENCES

1. Collins, G. E.; and Loos, R.: Polynomial Real Root Isolation by Differentiation. Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation, Aug. 1976, pp. 15-25.

2. Adams, D. A.: A Stopping Criterion for Polynomial Root Finding. Commun. ACM 10, no. 10, Oct. 1967, pp. 655-658.

3. Kahan, W.: Implementation of Algorithms, Parts I and II, Tech. Rep. 20, Dept. of Computer Science, University of California (Berkeley), 1973. (Available from DDC as AD 769 24.)

4. Fateman, R.: The MACSYMA 'Big Floating-Point' Arithmetic System. Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation, Aug. 1976, pp. 209-213.

5. Smith, B. T.: Error Bounds for Zeros of a Polynomial Based Upon Gerschgorin's Theorems. J. Assoc. Comput. Mach., vol. 17, no. 4 (Oct. 1970), pp. 661-674.

6. Pinkert, J. R.: An Exact Method for Finding the Roots of a Complex Polynomial. ACM Trans. on Math. Software, vol. 2, no. 4 (Dec. 1976), pp. 351-363.

Table 1: Time for Finding the Isolation Intervals for the nth Legendre Polynomial

| n | SAC-1 UNIVAC | | PDP-10 TRANSLATION | | FLOATING POINT | | | FLT. PT/INT. ARITH | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | time* | eevs | time* | eevs | time* | fevs | eevs | time* | fevs | eevs |
| 5 | 0.5 | 39 | 0.131 | 46 | 0.129 | 45 | 7 | 0.083 | 49 | 3 |
| 10 | 2.2 | 149 | 0.529 | 163 | 0.485 | 162 | 9 | 0.273 | 166 | 5 |
| 15 | 5.6 | 320 | 1.44 | 357 | 1.15 | 356 | 12 | 0.597 | 360 | 8 |
| 20 | 11.9 | 555 | 3.11 | 598 | 2.16 | 597 | 16 | 1.03 | 601 | 12 |
| 25 | 23.2 | 877 | 6.29 | 945 | 4.14 | 944 | 63 | 2.81 | 856 | 117 |
| 30 | ? | ? | 10.9 | 1326 | 7.57 | 1325 | 195 | 7.31 | 875 | 388 |

*Multiply time in seconds by 1.5 to include storage reclamation time. This ratio has been estimated for SAC-1, and is typical for MACSYMA measurements (although actual time is highly dependent on amount of system free storage). The column labelled "eevs" indicates number of exact evaluations, "fevs" floating point evaluations.

The first column (n) is the degree. The next two columns are derived from reference 1. The 4th and 5th columns are times and counts for the PDP-10 translation of the Collins-Loos algorithm. The 6,7, and 8 columns detail the results of using floating point evaluations, then exact evaluation. The last three columns indicate the results when floating point, and floating point interval arithmetic were used. The results in the last column could be improved for high degree polynomials by attempting operations in floating point rather than giving up on a complete stage when an overflow or insufficiently precise result is encountered.

BLANK PAGE

BLANK PAGE

5

# FLOATING POINT ISOLATION OF ZEROS OF A REAL POLYNOMIAL VIA MACSYMA

Bruce W. Char
University of California, Berkeley

## ABSTRACT

Given a square-free polynomial P of degree n with floating-point representable (real) coefficients, we would like to find n disjoint regions, each containing a root of P. Existing methods (ref. 1, 2) can be slow because of their reliance upon rational arithmetic. We propose a faster technique which uses only floating point arithmetic. A MACSYMA function, BOUND, was written which when given such a polynomial P, produces n complex discs C[i], each containing a true root of P. After computing the discs, BOUND determines if they are a set of isolating regions for the true roots of P (i.e. that no two of the C[i] overlap). The routine uses the Jenkins-Traub zero-finding algorithm (ref. 3)— MACSYMA's ALLROOTS function— to get approximations to the zeros, each approximation becoming the center of a disc. The radius of each C[i] is based upon error bound results by Adams (ref. 4) and Smith (ref. 5).

BOUND runs in time $O(n^2)$, with all calculations using the standard floating-point arithmetic of the Decsystem-10. As a compiled MACLISP routine, BOUND has been found to be 10 to 100 times faster than rational arithmetic root-isolating techniques in SAC-1 on the Univac 1110 and the Decsystem-10 by Pinkert (ref. 1) and Collins and Akritas (ref. 2), on test polynomials of degree 15 or less. It should be noted however, that BOUND does not allow the user to specify the size of the zero-containing regions nor is it guaranteed to find isolating regions as the rational arithmetic methods are. It may also break down due to underflow/overflow during intermediate computations on ill-conditioned polynomials. A technique to extract the best of both the rational and floating-point arithmetic approaches would be to use the above procedure as a quick first attempt, reserving rational arithmetic for when the initial method fails.

We anticipate several developments that will improve or extend BOUND. Since the Jenkins-Traub algorithm and Adams's and Smith's results work for polynomials with complex coefficients, the addition of complex arithmetic to MACLISP will allow BOUND to be easily extended to work in that general case. Because the quality of the zerofinding and the radii of the C[i] are in part dependent on the precision of the floating-point representation, BOUND would produce smaller regions if implemented in double precision. Still being investigated are the improvement of the existing error bounds, and development of methods that can be applied to polynomials with non-rational coefficients.

# REFERENCES

1. Pinkert, James R.: An Exact Method for Finding the Roots of a Complex Polynomial. ACM Transactions on Mathematical Software, vol. 2, no. 4, Dec. 1976, pp. 351-363.

2. Collins, George E.; and Akritas, Alkiviadis: Polynomial Real Root Isolation Using Descarte's Rule of Signs. Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation. Association of Computing Machinery, 1976. pp. 272-275.

3. Jenkins, M.A.; and Traub, J.F.: Zeros of a Complex Polynomial. Communications of the ACM, vol. 15, no. 2, Feb. 1972, pp. 97-99.

4. Adams, Duane A.: A Stopping Criterion for Polynomial Root Finding. Communications of the ACM, vol. 10, no. 10, Oct. 1967, pp. 655-658.

5. Smith, Brian T.: Error Bounds for Zeros of a Polynomial Based Upon Gerschgorin's Theorems. Journal of the Association for Computing Machinery, vol. 17, no. 4, Oct. 1970, pp. 661-674.

# PRESERVING SPARSENESS IN MULTIVARIATE POLYNOMIAL FACTORIZATION

Paul S. Wang
Laboratory for Computer Science & Mathematics Department, MIT

## INTRODUCTION

Working on heuristic programs for factoring polynomials over the integers, Claybrook has come up with many fairly large multivariate polynomials. He has proposed ten of these polynomials as test cases for any algorithmic approach to factoring (ref. 1). Attempts were made to factor these ten polynomials on MACSYMA (ref. 2). However it did not get very far with any of the larger polynomials. At that time MACSYMA used an algorithm created by Wang and Rothschild. This factoring algorithm has also been implemented for the symbolic manipulation system, SCRATCHPAD (ref. 3) of IBM. A closer look at this old factoring algorithm (OFA) (ref. 4) revealed three problem areas, each of which contribute to losing sparseness and intermediate expression growth. This study led to effective ways of avoiding these problems and actually to a new factoring algorithm (NFA) (ref. 5), (ref 6).

The three problems are known as the extraneous factor problem, the leading coefficient problem, and the bad-zero problem. These problems are examined separately in the following three sections. Their causes and effects are set forth in detail. Then the ways to avoid or lessen these problems are described.

The NFA has been implemented on MACSYMA. Its performance on the ten polynomials proposed by Claybrook is tabulated in Appendix A.

## AVOIDING EXTRANEOUS FACTORS

Consider factoring $U(x, x_2, ..., x_t) \in Z[x, x_2, ..., x_t]$ which is primitive and squarefree. U is reduced to a polynomial with only one variable by substituting selected integers for $x_2, ..., x_t$. Let $\tilde{U}(x) = U(x, a_2, ..., a_t)$. Factors of U are constructed from the irreducible factors of $\tilde{U}(x)$ by a kind of Hensel process.

An extraneous factor in this context is a univariate factor of U(x) over Z which does not lead to an actual factor of $U(x, ..., x_t)$, after multivariate p-adic construction. Consider, for example,

$$U(x,y,z) = (x^3 + y^4z^3)$$

If the evaluation $y = z = 1$ is made, then

$$U = (x, 1, 1) = (x^3 + 1) = (x^2 - x + 1)(x + 1).$$

Since $U(x,y,z)$ is irreducible over Z, neither of the two univariate factors can lead to a real factor of $U(x,y,z)$. They are all extraneous factors.

Obviously the cause of getting extraneous factors is unlucky points of evaluation. There are three undesirable effects of having such factors in the factoring process. Firstly, a combinatorial search for true factors has to be done at the end of the factoring procedure. Secondly, the multivariate p-adic construction often has to be carried out all the way to reach the bound for the total degree, $h$, of $U(x, x_2, \ldots, x_t)$ in $x_2, \ldots, x_t$, as opposed to reaching $\lceil h/r \rceil$ on the average, if all $r$ factors are not extraneous. Thirdly, the extraneous factors grow in size and density as they go through the multivariate construction process, quite uninhibited by the size or density of the given polynomial.

To illustrate the growth phenomenon, let us continue the example where $F_0(x) = x^2 - x + 1$, $G_0(x) = x + 1$ and

$$U(x,y,z) = x^3 + y^4 z^3 \equiv F_0(x)\, G_0(x) \mod \underline{s}$$

where $\underline{s}$ is the ideal $(y-1, z-1)$.

The multivariate p-adic construction produces from $F_0$ and $G_0$ polynomials $F_i$ and $G_i$ such that

$$U \equiv F_i G_i \quad \mod \quad (\underline{s}^{i+1}, b)$$

where $b$ is a prime or prime power bigger than the coefficient bound.

The first few $F_i$ and $G_i$ are shown below with $b=625$.

$$F_1 = 2Z + X\,(-Z + 207Y - 1) + 211Y + x^2 + 1$$
$$G_1 = Z - 207Y + X + 1$$

$$F_2 = 2Z + X(-Z + 207Y - 1) + 211Y + X^2 + 1$$
$$G_2 = Z - 207Y + X + 1$$

$$F_3 = Z^2 + X((207Y - 1)Z - 278Y^2 + 207Y - 1)$$
$$+ (2 - 203Y)Z + 280Y^2 + 211Y + X^2 + 1$$

$$G_3 = (1 - 207Y)Z + 278Y^2 - 207Y + X + 1$$

Therefore it is clear that extraneous factors should be avoided if at all possible. The approach taken here is to evaluate the given polynomial $U(x, \ldots, x_t)$ at several different sets of points $\{a_2, \ldots, a_t\}$ and to factor these resulting univariate images over Z. The set that gives the minimum number of factors will be selected. This means that the requirement in OFA of getting many zeroes and plus- or minus-one's as substitution values has to be relaxed. For the purpose of avoiding extraneous factors the conditions on the $a_i$'s are: (1) $\deg\tilde{U}(x) = \deg U(x,\ldots,x_t)$ in $x$ and (2) $\tilde{U}(x)$ is squarefree. If these $a_i$'s are generated at random, then the probability of getting an extraneous factor for any one set of $a_i$'s is low.

To use several different substitutions and choose the best should virtually eliminate the possibility of the occurrence of extraneous factors. Experiments on the machine indicate that two to three different substitutions will almost always suffice. Furthermore, the different univariate factorizations can be matched for degree compatibility among the factors. This, of course, provides additional information on the number of true factors.

Although one would like to use random evaluations, one would also like to use integers that are small in size so that the coefficients of $u(x)$ are not unnecessarily large. In the program, the substitution sets are generated randomly modulo a prime which is increased in size for each new set.

## SOLVING THE LEADING COEFFICIENT PROBLEM

The given polynomial $U(x,\ldots,x_t)$ can be written for a selected main variable, say $x$, in the form

$$U = V_n x^n + \ldots + V_0$$

where $V_i \in Z[x_2, \ldots, x_t]$. $V_n \neq 0$ is the leading coefficient. In this paper, the term "leading coefficient" always means that of the main variable, $x$. Some older factoring algorithms, for example, (ref. 7), require a monic input. If $V_n \neq 1$ then the change of variable $x = y/V_n$ is made and the monic polynomial

$$W = V_n^{n-1} U\left(\frac{y}{V_n}, x_2, \ldots, x_t\right)$$

is factored. An inverse transformation is required on the irreducible factors thus obtained. This approach is impractical because coefficients of W are much larger and denser than those of U. In **OFA** no such monic transformation is made. Instead, a leading coefficient recovery scheme is used.

In the multivariate case, the leading coefficient problem is caused by $V_n$ not being an integer. Let $f(x) = (x^2 + 1)$, $g(x) = (x^2 + x + 1)$ and $\tilde{U} = f(x)g(x)$ over Z. In doing the multivariate p-adic construction one computes the difference

$$R(x, \ldots, x_t) = f(x)g(x) - U(x, \ldots, x_t)$$

If $V_4$ is not an integer, then degree of R in $x$ is 4, which is the degree of U in $x$. This means for example one may get something like $c(x) = 3x^4 + 2x$ as the coefficient for, say, the $(x_2 - a_2)$ term in R. And the following congruence has to be solved

$$\alpha(x)f(x) + \beta(x)g(x) = 3x^4 + 2x. \qquad (1)$$

If $\deg(c(x)) < \deg(f) + \deg(g)$, there exist unique $\alpha$ and $\beta$ with $\deg(\alpha) < \deg(g)$ and $\deg(\beta) < \deg(f)$ satisfying $\alpha f + \beta g = c$. However, this is not the case for equation (1). In fact, one has

$$-5f + (3x^2 - 3x + 5)g = c(x)$$
$$(3x^2 + 3x - 2)f + (-3x + 2)g = c(x)$$

and an infinite number of linear combinations of these two equations. Because $\alpha(x)$ and $\beta(x)$ are used to correct the factors and because the true factors and their homomorphic images are unique, complications arise if $\alpha$ and $\beta$ are nonunique. In OFA a unique selection is made based on the condition $\deg(\alpha) \le \deg(g)$, $\deg(\beta) < \deg(f)$. However this choice can not be more appropriate than the condition $\deg(\alpha) < \deg(g)$ and $\deg(\beta) \le \deg(f)$. In either case, the factors thus constructed are only correct up to units in the underlying coefficient domain of truncated p-adic polynomials in $x_2,\ldots x_t$. Therefore they often are much denser than necessary. This also explains why correct coefficients have to be recovered after the p-adic construction.

Dealing with the leading coefficient problem in the context of the polynomial greatest common divisor computation, Yun (ref. 8) suggested that the leading coefficients of the given polynomial or an easily computible divisor of it be "imposed" on the univariate factors for p-adic construction. The solution to the leading coefficient problem here is to "predetermine" the correct leading coefficients of the factors of $U(x,\ldots,x_t)$.

To do this, the leading coefficient of $U(x,\ldots,x_t)$, $V_n$, is factored over Z first. Let

$$V_n = F_1^{e_1} F_2^{e_2} \ \ldots \ F_k^{e_k}$$

where $F_i$ are distinct irreducible polynomials in $Z[x_2,\ldots,x_t]$. Some of the $F_i$'s may be integers. Let us assume that $V_n \ne$ an integer, for the case is trivial otherwise. Let $\tilde{F}_i = F_i(a_2,\ldots,a_t)$. The integers $\{a_2,\ldots,a_t\}$ are chosen to satisfy the two conditions given in the previous section, and, for leading coefficient distribution, the additional condition: For each nonintegral $F_i$, $\tilde{F}_i$ has at least one prime divisor $p_i$ which does not divide any $\tilde{F}_j$, $j \ne i$, or the content of $\tilde{U}(x)$.

Let $\sigma$ be the content of $\tilde{U}(x)$ and $u(x) = \tilde{U}/\sigma$. Now $u(x)$ can be factored into distinct irreducible factors over Z.

$$u(x) = u_1(x) \ldots u_r(x).$$

Assuming no extraneous factors, then $U(x,\ldots,x_t)$ has r distinct irreducible factors $G_i(x,\ldots,x_t)$, $i = 1,\ldots,r$. Let $C_i(x_2,\ldots,x_t)$ be the leading coefficient of $G_i$, $\tilde{C}_i = C_i(a_2,\ldots,a_t)$ and $G_i(x,a_2,\ldots,a_t) = \sigma_i u_i(x)$ where $\sigma_i$ is some divisor of $\sigma$. The following lemma allows one to determine $C_i(x_2,\ldots,x_t)$ up to integer multiples.

Lemma If there are no extraneous factors then, for all i,j and m, $F_j^m$ divides $C_i$ if and only if $\tilde{F}_j^m$ divides $lc(u_i)\sigma$.

Proof If $F_j^m | C_i$ then $\widetilde{F}_j^m$ divides $\widetilde{C}_i = lc(u_i)\sigma_i$. On the other hand, if $F_j^m$ does not divide $C_i$ then $\widetilde{C}_i = \widetilde{F}_1^{s_1} \cdots \widetilde{F}_k^{s_k}$ with $s_j < m$. Thus $p_j^m$ does not divide $\widetilde{C}_i$ which implies that $\widetilde{F}_j^m$ does not divide $lc(u_i)\sigma$.

The readers are referred to [6] for details of this leading coefficient distribution algorithm The process will be illustrated here by an example. Consider

$$U(x,y,z) = ((y^2-z^2)x^2 + y - z^2) * (4(y + z)x^2 + xyz - 1) * (yz^2x^2 + 3xz + 2y),$$

where the factors are to be found. Factoring the leading coefficient of $U(x,y,z)$ over Z gives

$$V_6 = 2^2yz^2(y + z)^2(y - z)$$

Therefore, we have $F_1=2$, $F_2=y$, $F_3=z$, $F_4=y + z$ and $F_5=y-z$. The sets of integers $\{5,-12\}$, $\{-14,3\}$ and $\{-23,3\}$ satisfy the three requirements and all give three factors for $U(x)$. Let us use $y= 5$ and $z= -12$. Thus $\widetilde{F}_1= 2$, $\widetilde{F}_2= 5$, $\widetilde{F}_3= -12$, $\widetilde{F}_4= -7$ and $\widetilde{F}_5= 17$. Factoring $\widetilde{U}(x) = U(x,5,-12)$ one obtains $\widetilde{U}(x) = 2u_1u_2u_3$

where

$$u_1 = 119x^2 + 139,$$

$$u_2 = 28x^2 + 60x + 1,$$

$$\text{and } u_3 = 360x^2 - 18x + 5.$$

Now $119 = -\widetilde{F}_4\widetilde{F}_5$ gives $C_1 = -F_4F_5 = (z^2 - y^2)$. Similarly, $C_2 = 4(y + z)$. And $2 * 360 = \widetilde{F}_2\widetilde{F}_3^2$ implies that $C_3 = yz^2$. These are correct leading coefficients of the true factors of $u(x,y,z)$ up to integer multiples.

## COMBATTING THE BAD ZERO PROBLEM

From $\widetilde{U}(x) = f(x)g(x)$ over Z with $f(x)$ and $g(x)$ relatively prime, the multivariate p-adic construction algorithm of **OFA** computes the difference

$$R(x,...x_t) = f(x)g(x) - U(x,...x_t)$$

which is congruent to zero mod $\underline{s}$, $\underline{s} = (x_2 - a_2,...x_t - a_t)$. Now R can be expressed in the form

$$R = c_2(x)(x_2 - a_2) + c_3(x)(x_3 - a_3) + ... + c_t(x)(x_t - a_t) + D(x,..,x_t).$$

where the $a_i$'s are the integers of evaluations and $D \equiv 0 \bmod \underline{s}^2$. The goal is to obtain the coefficients $c_2(x),...,c_t(x)$. In other words, we need the coefficients of the linear terms in the power series expansion of P at $x_2 = a_2,\dots,x_t = a_t$. In general, for the stage of the p-adic construction where the residue is zero mod $\underline{s}^i$ but nonzero mod $\underline{s}^{i+1}$, the coefficients of the degree $i$ terms in the power series form of R will be needed. One way to do this is to substitute $y_i + a_i$ for $x_i$ and work with $U(x, y_2 + a_2, \dots, y_t + a_t)$ expanded. After the substitution, $\underline{s}$ becomes $(y_2,\dots,y_t)$ and obtaining coefficients of terms in $y_2,\dots,y_t$ of any degree is very easy. Furthermore modulo operations with $\underline{s}^i$ are simply truncations.

However substitution and expansion greatly increase the size and density of U. For instance, a term $x_2^a x_3^b x_4^c$ becomes $(y_2 + a_2)^a (y_3 + a_3)^b (y_4 + a_4)^c$ which has $(a + 1)(b + 1)(c + 1)$ terms when expanded. The exponential growth is worst if all $a_i$'s are not zero. Hence the name "bad-zero problem." This growth problem is so bad that the factoring program may run out of core for moderately-sized polynomials.

Therefore, such substitution should not be made. If $R \equiv 0 \bmod \underline{s}^i$, and $R \not\equiv 0 \bmod \underline{s}^{i+1}$, then the coefficient of $(x_2 - a_2)^i$, for example, can be obtained by the formula

$$\frac{1}{i!}\left(\frac{d^i}{dx_2^i} R(x, x_2, \dots, x_t)\right)_{x_2=a_2}$$

A typical term of degree $i$ in $R(x,\dots,x_t)$ looks like

$$c(x)\ (x_2-a_2)^{e_2} \dots (x_t - a_t)^{e_t},\quad e_1+\dots+e_t = i. \qquad (2)$$

To obtain c(x) one uses the general formula

$$\frac{1}{e_2! \dots e_t!} \frac{d}{dx_2}^{e_2} \dots \frac{d}{dx_t}^{e_t} R(x, \dots, x_t) \Bigg|_{x_i = a_i} \qquad (3)$$

This method has no exponential expression growth problem. Polynomial differentiation and evaluation being relatively inexpensive, it should be an improvement over the **OFA** which uses substitution and expansion. Many polynomials that can not be factored by **OFA** because of storage problems should be doable by this method. However, the number of possible terms in the form (2) can be large, which means (3) may be computed many times.

In the worst case, $i$ equals $h$, which is the total degree of $U(x, x_2, \dots x_t)$ in $x_2, \dots x_t$. The number of possible terms in the form (2) with $e_2 + \dots + e_t = h$ is then given by $\binom{h + t - 2}{t - 2}$ which is of order $o(h^{t-2})$ if $h$ is much larger than $t$. However if there are no extraneous factors and if the leading coefficients of the factors are correctly determined, then (i) the maximum degree of any $x_i$, $i = 2,\dots,t$ in the factors are much less than $h$ and (ii) the p-adic construction often need only be carried out to $i = [h/r]$ if there are $r$ factors. Even so, experiments on the machine indicate that many applications of formula (3) result in zero. In other words, too often we are looking for terms that are not there. The way to improve the situation is to do the p-adic construction variable-by-variable instead of introducing all variables $x_2,...,x_t$ at once. Thus the actual factors of $U(x,x_2,a_3,...,a_t)$ are constructed first. From these factors in two variables, the true factors of $U(x,x_2,x_3,a_4,...,a_t)$ are then constructed, etc. We shall not go into details here. Interested readers are referred to [6] where a linearly convergent variable-by-variable parallel p-adic construction is described in full detail.

The author wishes to thank Joel Moses for suggesting this paper and Miss Dianne Foster for careful copying and editing.

## REFERENCES

1. Claybrook, B.G.: Factorization of Multivariate Polynomials Over the Integers. SIGSAM Bulletin, Feb. 1976, p. 13.

2. The Mathlab Group: MACSYMA Reference Manual. Lab. for Comp. Sci., Massachusetts Inst. Technol., Nov. 1975.

3. Griesmer, J.H.; Jenks, D.R.; and Yun, D.Y.Y.: SCRATCHPAD Users' Manual. IBM Research Report, RA 70, June 1975.

4. Wang, P.S.; and Rothschild, L.P.: Factoring Multivariate Polynomials Over the Integers. Math. Comp., vol. 29, no. 131, July 1975, pp. 935-950.

5. Wang, P.S.: Factoring Larger Multivariate Polynomials. SIGSAM Bulletin, Nov. 1976.

6. Wang, P.S.: An Improved Multivariate Polynomial Factoring Algorithm. Submitted to Math. Comp., 1977.

7. Musser, D.R.: Algorithms for Polynomial Factorization. PhD Thesis, Univ. of Wisconsin, Aug. 1971.

8. Yun, D.Y.Y.: The Hensel Lemma in Algebraic Manipulation. PhD Thesis, Massachusetts Inst. Technol., Nov. 1973.

# APPENDIX A

Contained here are ten factoring examples done by MACSYMA using the old factoring algorithm (OFA) (ref. 4) and the new factoring algorithm (NFA) (ref. 6). These polynomials are proposed by Claybrook (ref. 1) who factored them using a heuristic approach. To conserve space, these polynomials are given in factored form below. The timing for OFA and NFA was done on a DEC KL-10. Claybrook's timings are obtained from (ref. 1). He did his timing on a Univac 1108. Times listed in Table 1 are in seconds. A * indicates running out of store.

FACTORING TIME COMPARISONS

| Polynomial | OFA | NFA | Claybrook |
|---|---|---|---|
| 1 | * | 3.30 | 174.66 |
| 2 | 0.96 | 0.95 | 6.85 |
| 3 | * | 7.83 | 10.06 |
| 4 | * | 5.12 | 149.26 |
| 5 | * | 9.07 | 160.03 |
| 6 | * | 5.92 | 172.16 |
| 7 | 0.27 | 0.28 | 1.97 |
| 8 | 3.398 | 0.58 | 25.38 |
| 9 | 10.52 | 2.82 | 67.49 |
| 10 | 79.68 | 0.58 | 129.01 |

TABLE 1

**The ten polynomials**

(1) $(W^4 Z^3 - X Y^2 Z^2 - W^4 X^5 Y^6 - W^2 X^3 Y)(- X^5 Z^3 + Y Z + X^2 Y^3)$

$(W^4 Z^6 + Y^2 Z^3 - W^2 X^2 Y^2 Z^2 + X^5 Z - X^4 Y^2 - W^3 X^3 Y)$

(2)
$$(Z + Y + X - 3)^{3} (Z + Y + X - 2)^{2}$$

(3)
$$\begin{aligned}&(- 15 Y^{2} Z^{16} + 29 W^{4} X^{12} Y^{12} Z^{3} + 21 X^{3} Z^{2} + 3 W^{15} Y^{20})\\
&(- Z^{31} - W^{12} Z^{20} + Y^{18} - Y^{14} + X^{2} Y^{2} + X^{21} + W^{2})\end{aligned}$$

(4)
$$\begin{aligned}&U^{4} X Z^{2} (6 W^{2} Y^{3} Z^{2} + 18 U^{2} W^{3} X Z^{2} + 15 U Z^{2} + 10 U^{2} W X Y^{3})\\
&(- 44 U W X Y^{4} Z^{4} - 25 U^{2} W^{3} Y Z^{4} + 8 U W X^{3} Z^{4} - 32 U^{2} W^{4} Y^{4} Z^{3}\\
&+ 48 U^{2} X^{2} Y^{3} Z^{3} - 12 Y^{3} Z^{2} + 2 U^{2} W X^{2} Y^{2} - 11 U W^{2} X^{3} Y - 4 W^{2} X)\end{aligned}$$

(5)
$$\begin{aligned}&(31 U^{2} X Z + 35 W^{2} Y^{2} + 6 X Y + 40 W X^{2}) (U^{2} W^{2} X Y^{2} Z^{2} + 24 U^{2} W X Y^{2} Z^{2}\\
&+ 12 U^{2} X Y^{2} Z^{2} + 24 U^{2} X^{2} Y Z^{2} + 43 W X Y Z^{2} + 31 W^{2} Y Z^{2} + 8 U^{2} W^{2} Z^{2}\\
&+ 44 U W^{2} Z^{2} + 37 U^{2} Y^{2} Z + 41 Y^{2} Z + 12 W X^{2} Y Z + 21 U^{2} W X Y Z + 23 X Y Z\\
&+ 47 U^{2} W^{2} Z + 13 U W^{2} X^{2} Y^{2} + 22 X Y^{2} + 42 U^{2} W^{2} Y^{2} + 29 W^{2} Y^{2} + 27 U W^{2} X^{2} Y\\
&+ 37 W^{2} X Z + 39 U W X Z + 43 U X^{2} Y + 24 X Y + 9 U^{2} W X^{2} + 22 U^{2} W^{2})\end{aligned}$$

(6)
$$\begin{aligned}&X Y (- 13 U^{3} W^{2} X Y Z^{3} + W^{3} Z^{3} + 4 U X Y^{2} + 47 X Y)\\
&(43 U X^{3} Y^{3} Z^{3} + 36 U^{2} W^{3} X Y Z^{3} + 14 W^{3} X^{3} Y^{3} Z^{2} - 29 W^{3} X Y^{3} Z^{2}\\
&- 20 U^{2} W^{2} X^{2} Y^{2} Z^{2} + 36 U^{2} W X Y^{3} Z - 48 U W X^{3} Y^{2} Z + 5 U W X^{2} Y^{3}\\
&+ 36 U W^{2} Y^{3} - 9 U W Y^{3} - 23 U W X^{3} Y^{2} + 46 U X^{3} Y^{2} + 8 X Y^{2} + 31 U^{2} W^{3} Y^{2}\\
&- 9 U^{2} Y^{2} + 45 X^{3} - 46 U^{2} W X)\end{aligned}$$

(7)
$$(Z + Y + X - 3)^{3}$$

(8) $$(3 Z^3 + 2 W Z - 9 Y^3 - Y^2 + 45 X^3) (W^2 Z^3 + 47 X Y - W^2)$$

(9) $$(- 18 X^4 Y^5 + 22 Y^5 - 26 X^3 Y^4 - 38 X^2 Y^4 + 29 X^2 Y^3 - 41 X^4 Y^2 + 37 X^4)$$

$$(33 X^5 Y^6 + 11 Y^2 + 35 X^3 Y - 22 X^4)$$

(10) $$X^6 Y^3 Z^2 (3 Z^3 + 2 W Z - 8 X Y^2 + 14 W^2 Y^2 - Y^2 + 18 X^3 Y)$$

$$(- 12 W^2 X Y Z^3 + W^2 Z^3 + 3 X Y^2 + 29 X - W^2)$$

# ON THE EQUIVALENCE OF POLYNOMIAL GCD AND SQUAREFREE FACTORIZATION PROBLEMS

**David Y. Y. Yun**

**Mathematical Sciences Department**
**IBM Thomas J. Watson Research Center**
**Yorktown Heights, New York, 10598 USA**

(Extended Abstract)

The importance of computing greatest common divisors (GCD's) of polynomials has been recognized more than a decade ago. All symbolic and algebraic computation systems must provide some form of polynomial GCD capability in order to handle the fundamental extension field of rational functions. The complexity of the GCD problem is aggravated by the fact that most of these systems use an expanded canonical representation for polynomials, which is at its worst, in terms of space requirement and comprehensibility, when the polynomials are multivariate. Much work has been done to understand and improve algorithms for computing GCD's over the past decade (ref. 1, 2, 3). But the need for a symbolic system to maintain relatively prime numerators and denominators in a rational function continues to cause a large amount of computer time to be spent computing GCD's.

In 1974, Brown (ref. 4) paved the way to a "factored" representation of rational functions for symbolic systems. The idea is that if both the numerator and denominator are factored into irreducible polynomials (primes in the polynomial domain) then the computation of GCD's simply involves finding the minimum powers of identical primes. Unfortunately, there are two drawbacks to Brown's approach. First, such a "factored" representation, though maintaining the relatively prime property of numerator and denominator (with minimum effort), does not result in canonically represented polynomials - that is, identical rational functions may appear differently in the numerator and denominator polynomials. The other is, as Brown correctly pointed out, factorization of polynomials into primes is too expensive an operation, so that his "factored" representation can only look for "sharable factors" by inexpensive means and maintain such partially factored forms. Consequently, equivalence of rational functions in such a representation can only be recognized by subtractions and, in most cases, expansions as well as GCD computations. Even though some symbolic systems have successfully utilized the "factored" representation (mainly in terms of the ability to comprehend expressions), it is not clear what is the actual trade-off between the effort for GCD computations that is presumably saved and the sacrifice of canonical form with the possible gain of maintaining some "sharable" factors.

In 1976, Yun published an improved algorithm for finding the "squarefree" factorization of a polynomial (ref. 5). By definition, a polynomial is said to be squarefree if it has no divisor (or factor) of multiplicity greater than 1. Thus, the problem of finding the squarefree factorization (abbreviated as SQFR) is that of finding polynomials

$P_1, P_2, \ldots, P_k$ such that $P = P_1{}^1 P_2{}^2 \ldots P_k{}^k$, where $P_k \neq 1$, each $P_i$ is squarefree, and $\gcd(P_i, P_j) = 1$ for all $i \neq j \leq k$.

Although the squarefree factorization is not quite the complete factorization of polynomials into primes, it is a canonical form for polynomials, as Yun pointed out. In fact, a result of Knuth indicates that the probability of the squarefree factorization being the same as the complete factorization for an arbitrary polynomial is approximately 4/5. Such a result further increases the usefulness of a squarefree representation for polynomials which has no parallel in the case of integers (i.e., given an integer, there is no known algorithm that will produce its squarefree factorization without finding its prime factorization first). On the other hand, squarefree factorization constitutes an essential step in polynomial factorization (ref. 6, 7, 8), partial fraction decomposition of rational functions (ref. 9), and rational function integration (ref. 10, 11, 12).

The mathematical theory for the new algorithm is given by the following three results (ref. 5):

Fundamental Theorem of Squarefree Decomposition:

> If $P(x)$ is a primitive polynomial in $D[x]$ where $D$ is a field of characteristic 0 and the squarefree factorization of $P$ is $P_1 P_2^2 \ldots P_k^k$, then $\gcd(P, P') = P_2 P_3^2 \ldots P_k^{k-2}$.

Corollary 1: Let $D = \gcd(P, P')$, then $P'/D - (P/D)' = P_1 \prod_{i=2}^{k} (i-1) P_i \prod_{j \neq i} P_j$.

Corollary 2: $\gcd(P/D, P'/D - (P/D)') = P_1$.

Based on these results, an algorithm for finding the squarefree factorization of a polynomial $P(x)$ can be given. Let $(G, A^*, B^*) \leftarrow \gcd(A,B)$ denote the computation of GCD of $A$ and $B$ and assignment of the GCD to $G$, $A/G$ to $A^*$, and $B/G$ to $B^*$.

Yun's algorithm (ref. 5) is as follows:

```
$(W, C_1, D_1) \leftarrow \gcd(P, P')$;
For $i = 1$, step 1, until $C_i = 1$,
    Do $(P_i, C_{i+1}, D_{i+1}) \leftarrow \gcd(C_i, D_i - C_i')$.
```

Yun's 1976 paper got as far as comparing three algorithms for squarefree factorization and showing the superiority of the new algorithm both experimentally and by algorithmic analysis of certain models for computation. However, there was no attempt to derive any specific expression for the computing cost bound nor any reducibility result. In this paper, we will show that the total computing cost of the squarefree factorization of a polynomial with degree n (i.e. SQFR(n)) is bounded by and, in fact, equal to 2*GCD(n). The crucial observation is that the inputs to calls of the GCD function in Yun's new algorithm are more "balanced" in terms of degrees than those algorithms previously proposed. Since the reduction of squarefree factorization problem to GCD problem hinges on the use of a two-argument function (GCD) to do the job of a one-argument function (SQFR), the balancing of degrees becomes especially important. (The other algorithms for squarefree factorization turn out to call on GCD functions with one input far more dominant in degree than the other.)

Thus, we will show that a closer re-examination of Yun's 1976 paper reveals the reducibility of SQFR to GCD. The natural question that follows is whether GCD is reducible to SQFR. That is answered affirmatively by the other half of this paper and the derivation will actually suggest an algorithm for computing GCD's when input polynomials are already represented by their SQFR form.

The fundamental theorem for this reduction process is

Theorem: For squarefree polynomials A and B,

$\mathrm{gcd}(A,B) = A{*}B/\mathrm{sqfrpt}(A{*}B)$

where the squarefree part of $P = \mathrm{sqfrpt}(P) = P_1P_2 \ldots P_k$,

if $P = P_1{}^1P_2{}^2 \ldots P_k{}^k$, hence, a by-product of sqfr(P).

This theorem, which is reminiscent to the relationship between GCD and LCM, suggests an obvious way of reducing GCD to SQFR. That is, for $F = F_1{}^1F_2{}^2 \ldots F_k{}^k$ and $G = G_1{}^1G_2{}^2 \ldots G_m{}^m$, compute $\mathrm{gcd}(F_i, G_j)$ for all i and j by the method of the theorem since each $F_i$ and $G_j$ is squarefree. (Note that this type of "cross GCDing" is also necessary for the "factored" representation of Brown.) Unfortunately, there are k*m GCD's required which forces k and m into the computing cost expression and affects the reduction process of GCD to SQFR — we are looking for strong reducibility of GCD to SQFR with constant cost for transformation of problems, as in the reduction of SQFR to GCD case where the constant is 2.

A corollary of the theorem provides a hint for a different approach.

**Corollary**: For polynomials F and G, let FS and GS denote sqfrpt(F) and sqfrpt(G) respectively.

Then $\mathrm{sqfrpt}(\mathrm{gcd}(F,G)) = \mathrm{gcd}(FS,GS) = FS{*}GS/\mathrm{sqfrpt}(FS{*}GS)$

Thus, a polynomial $D_1 = \mathrm{sqfrpt}(\mathrm{gcd}(F,G))$ can be computed, according to the corollary, from $\mathrm{sqfrpt}(F) = F_1F_2 \ldots F_k$ and $\mathrm{sqfrpt}(G) = G_1G_2 \ldots G_m$.

Similarly, we compute $D_j = \mathrm{gcd}(F_j \ldots F_k, G_j \ldots G_m)$ according to the corollary for all j up to min(k,l). Finally, it will be shown that $\mathrm{gcd}(F,G) = D_1D_2 \ldots D_{\min(k,m)}$.

The important technique in this case is "triangularization". As opposed to the k*m cross GCD's, squarefree parts of F and G are peeled off successively and collectively. The total cost of computing the D's, hence the GCD, via the method of the corollary adds up to less than 6*SQFR(n), where the degrees of F and G are assumed to be n. In other words, GCD(n) problem is strongly reducible to SQFR(n) with a multiplying constant of 6.

If F and G are already in SQFR form, then the cost for computing their GCD is bounded by 4*SQFR(n), i.e., the cost for computing GCD of polynomials in SQFR form is not more than twice that of putting them in SQFR form originally. Another potential advantage of such a GCD algorithm is that the computing cost will be generally dependent on the minimum of the degrees of the input polynomials when the degrees are not equal, mainly because the computation goes on only until min(k,m) is reached. Previously, all GCD algorithms have shown a strong dependence on the maximum of the degrees, which is the cause of the need to "balance" the inputs of calls to GCD functions, as noted earlier.

At this point, we can draw the following conclusion

Theorem: GCD(n) problem is equivalent to SQFR(n) problem.

It should be noted that the derivation of above results are based on the assumptions that

$$a^2 M(n) \geq M(a\, n) \geq a\, M(n) \text{ for all } a \geq 1$$

where M(n) stands for the cost for multiplying polynomials of degree n (ref. 13, p. 280). Let X(n) denote M(n), GCD(n), or SQFR(n). Then the satisfiability of the following condition has also been assumed:

$$\sum_{i=1}^{k} X(n_i) \leq X(\sum_{i=1}^{k} n_i) \quad \text{for any } n_i \text{ in } N.$$

We point out, however, this condition is easily satisfied by the above operation costs, so that it represents no severe restriction on our result.

## REFERENCES

1. Brown, W. S.: On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisor. JACM, vol. 18, no. 4, Oct. 1971, pp 478-504.

2. Moses, J.; and Yun, D. Y. Y.: The EZ GCD Algorithm. Proceedings of ACM Annual Conference, Aug. 1973, Atlanta, pp. 159-166.

3. Moenck, R.: Fast Computation of GCD's. Proceedings of 5th Annual ACM Symposium on Theory of Computing, 1973, 142-151.

4. Brown, W. S.: On Computing with Factored Rational Expressions. Proceedings of EUROSAM '74, also SIGSAM Bulletin No. 31, Aug. 1974, pp. 26-34.

5. Yun, D. Y. Y.: On Square-free Decomposition Algorithms. Proceedings of 1976 ACM Symposium on Symbolic and Algebraic Computation, R. D. Jenks, ed., Aug. 1976, pp. 26-35.

6. Knuth, D.: The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, Addison-Wesley, Reading, Mass., 1969.

7. Musser, D. R.: Algorithms for Polynomial Factorization, Ph. D. Thesis, C. S. Dept., Univ. of Wisconsin, August 1971.

8. Wang, P. S.; and Rothschild, L. P.: Factoring Multivariate Polynomials Over the Integers. Mathematica of Computation, Vol. 29, No. 131, July 1975, pp. 1035-950.

9. Kung, H. T.; and Tong, D. M.: Fast Algorithms for Partial Fraction Decomposition. Dept. of Computer Science Report, Carnegie-Mellon University, January 1976.

10. Moses, J.: Symbolic Integration: The Stormy Decade. Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, S. R. Petrick, ed., March 1971, 427-440.

11. Horowitz, E.: Algorithms for Partial Fraction Decomposition and Rational Function Integration. Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, S. R. Petrick, ed., March 1971, 441-457.

12. Yun, D. Y. Y.: Fast Algorithm For Rational Function Integration. Proceedings of IFIP Congress 77, Aug. 1977, Toronto, Canada.

13. Aho, A. V.; Hopcroft, J. E.; and Ullman, J. D.: The Design and Analysis of Computer Algorithms, Addison-Wesley, Mass., 1974.

# DIFFERENTIAL FORM ANALYSIS USING MACSYMA*

Hugo D. Wahlquist
Jet Propulsion Laboratory
California Institute of Technology

## ABSTRACT

The calculus of exterior differential forms has increasing applications in several areas of applied mathematics and theoretical physics. The formalism was developed initially by E. Cartan (ref. 1) for his own research in differential geometry. Modernized and updated by present day mathematicians, it has become a standard tool for mathematical work in the differential geometry of manifolds (refs. 2 and 3).

With that genesis it is not surprising that the techniques of differential forms are useful in general relativity (ref. 4). Many problems in relativity can be concisely expressed and efficiently solved using differential forms together with Cartan's "method of moving frames." The calculational effort involved is often significantly reduced compared to the standard tensor formalism. Other areas of theoretical physics in which differential forms have utility, as well as elegance, include Hamiltonian mechanics, statistical mechanics, and the calculus of variations (refs. 5 and 6).

In recent years the geometric techniques of exterior calculus developed (again by Cartan) for systems of partial differential equations (refs. 1 and 7) have been applied to physically important nonlinear equations. Many results on transformation properties, invariance groups, and conservation laws can be derived directly and systematically using these methods (ref. 8). When the methods are applied to nonlinear equations which exhibit the recently discovered "soliton" phenomenon (the Korteweg-de Vries equation, for instance), a beautiful algebraic structure associated with the equations is revealed. These so-called "prolongation structures," which are essentially "free" Lie algebras, can be shown to lead directly to solution methods such as the inverse scattering method, Bäcklund transformations, and exact nonlinear superposition principles (ref. 9). The prolongation structures also have a geometrical interpretation in terms of affine connections over solution manifolds (ref. 10). From this viewpoint they appear to be closely related to nonlinear, gauge-invariant, field theories; the Yang-Mills fields.

---

*This paper presents the results of one phase of research carried out at the Jet Propulsion Laboratory, California Institute of Technology, under Contract No. NAS7-100, sponsored by the National Aeronautics and Space Administration.

The utility of differential forms is not limited to proving abstract general theorems; they also provide an efficient calculational tool for deriving particular results in specific problems (ref. 11). As in other areas of analysis, the computer can be of great help in carrying out the actual manipulations. Exterior calculus has been implemented in Pl/1-FORMAC by F. Ernst (ref. 12). The major purpose of his programs was to facilitate the use of differential forms in general relativity, although the programs are not restricted to that application. Recently, we have written a small file of routines in MACSYMA which we are using to perform differential form calculations in the theory of nonlinear differential equations. These routines accomplish only partial implementation; in fact, the main reason for this paper is to advertise the need for implementing exterior calculus in MACSYMA which clearly has the facilities to do the complete job. My hope is to provoke enough interest in someone sufficiently knowledgeable to do the job right.

Algebraically, the differential forms constitute a Grassman algebra over the cotangent space of a manifold involving the noncommutative exterior product operation, usually denoted by the wedge symbol, $\wedge$. The exterior derivative, d, is the unique operation of differentiation leading from one differential form to another. Its application to a form of rank p results in a form of rank $p + 1$.

When in addition the dual tangent vectors of the manifold are introduced, new invariant algebraic and derivative operations can be defined: contraction between vectors and forms, and Lie derivatives of both forms and vectors.

The paper describes the MACSYMA file which has been written to perform these operations and discusses the improvements and additions which are needed to accomplish a complete and efficient implementation. Examples of differential form calculations are also displayed.

## REFERENCES

1. Cartan, E., "Les systèmes differentials extèrieurs et leurs applications géométrique," Hermann, Paris, 1945. This is the principal source.

2. Hermann, R., *Interdisciplinary Mathematics*, Vols. I-XII, Math. Sci. Press, Brookline, MA, 1976. Much of the classical work on differential geometry, partial differential equations, Lie groups, etc. which is important for applications is reconstructed in these volumes in modern mathematical language and notation, often employing exterior calculus.

3. Choquet-Bruhat, Y., "Géométrie différentielle et systèmes extérieurs," Dunod, Paris, 1968. A thorough, fairly rigorous, well-balanced coverage of exterior calculus.

4. Israel, W., "Differential forms in general relativity," Comm. Dublin Inst. Adv. Studies Ser. A, No. 19, 1970. The utility of differential forms in general relativity is convincingly demonstrated herein.

5. Flanders, H., *Differential Forms*, Academic Press, New York, 1963; D. Lovelock, and H. Rund, *Tensors, Differential Forms, and Variational Principles*, Wiley, New York, 1975. These are both excellent introductions to the subject of differential forms with applications to Riemannian geometry, variational calculus, and other areas of theoretical physics.

6. Estabrook, F. B. and Wahlquist, H. D., "The geometric approach to sets of ordinary differential equations and Hamiltonian dynamics," SIAM Review 17, No. 2, 201-220, 1975. A review article treating Hamiltonian theory in the language of differential forms.

7. Slebodzinski, W., *Exterior Forms and Applications*, Polish Scientific Publ., Warsaw, 1970. A fairly complete summary of Cartan's theory of partial differential systems is included. In English translation.

8. Harrison, B. K. and Estabrook, F. B., "Geometric approach to invariance groups and solutions of partial differential systems," J. Math. Phys. 12, 653-666, 1971. The Lie derivative of the differential ideal of forms is used to systematize invariance groups and similarity solutions of partial differential equations.

9. Wahlquist, H. D. and Estabrook, F. B., "Prolongation structures of nonlinear evolution equations," J. Math. Phys. 16, 1-7, 1975. Also, Estabrook, F. B. and Wahlquist, H. D., II, J. Math. Phys. 17, 1293-1297, 1976. Two papers with identical titles which introduce the concept of prolongation structures, derive them for particular nonlinear soliton equations, and relate them to solution methods.

10. Hermann, R., "The pseudopotentials of Estabrook and Wahlquist, the geometry of solitons, and the theory of connections," Phys. Rev. Lett. 36, 835, 1975. In which the geometric significance of prolongation structures is revealed.

11. Estabrook, F. B., "Some old and new techniques for the practical use of exterior differential forms," in Bäcklund Transformations, No. 515, Springer Lecture Notes in Mathematics, p. 136-161, Ed. R. M. Miura, Springer-Verlag, New York, 1976. A concise discussion of forms and their uses together with properties, identities, and calculational techniques.

12. Ernst, F., "Manipulation of differential forms on a digital computer," Proceedings of the Relativity Seminar, PORS IIT-10, Illinois Institute of Technology, Chicago, 1969. A documentation of the PL/1-FORMAC implementation of exterior calculus.

# INDICIAL TENSOR MANIPULATION ON MACSYMA

Richard A. Bogen
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Richard Pavelle
Logicon, Inc.
Hartwell Avenue
Lexington, Massachusetts 02173

## ABSTRACT

We describe a new computational tool for physical calculations. It is the first computer system capable of performing indicial tensor calculus(as opposed to component tensor calculus). It is now operational on the symbolic manipulation system MACSYMA. We outline the capabilities of the system and describe some of the physical problems we have considered as well as others we are examining at this time.

## INTRODUCTION

Symbolic or algebraic computer manipulation systems are finding a growing role in physics by performing complex calculations without error. While symbolic manipulation has been used in Quantum Electrodynamics, Quantum Mechanics, Celestial Mechanics and gravitation theories (ref.1) , it is in the gravitation theories where these systems are now becoming essential tools. Symbolic manipulation gives one the ability to guess at exact solutions of gravitational field equations or use approximation procedures to find them (ref.2). Symbolic calculation also provides one the freedom to consider lengthy problems whose solution by hand would be error prone and could take months. A recent paper reviews some of the problems in gravitation which have been studied using symbolic manipulation as well as the computing systems which are now in use (ref.3).

The usual symbolic computing system for gravitation calculations operates in the following manner: The user often wishes to study a particular metric and inputs each specific component relative to a coordinate system or noncoordinate frame. The system then computes the geometric objects or differential equations of interest. There are many types of relativistic calculations which computer systems are performing (ref.3). We have had such a system running on MACSYMA since 1973. In 1974, however, we began construction of a novel package for performing actual indicial tensor analysis as opposed to the usual component tensor calculus. The purpose of this paper is to describe the current capabilities of our indicial tensor manipulation system, ITMS. We shall also describe some of the problems we have solved as well as others of current interest.

## INDICIAL TENSOR MANIPULATION

We represent a tensor $T^{ijk..}_{rs...}$ as a function of two arguments which are the lists of indices. A list in MACSYMA is a sequence of its elements which are separated by commas and enclosed by square brackets. Thus we write the above tensor as T([r,s,...],[i,j,k,..]) while a scalar is represented by a function with empty lists such as P([],[]).

In ITMS ordinary differentiation of a tensor with respect to a coordinate $x^k$ causes the k index to be appended onto the list(tensor) as an additional argument to the tensor function. Thus we represent $T_{ij,k}$ as T([i,j],[],k). Since ordinary differentiation is commutative, multiple ordinary derivative indices are sorted in alphanumeric order causing expressions such as T([i,j],[],k,n,m) - T([i,j],[],m,k,n) to vanish automatically as part of MACSYMA's simplification routine. We may also declare a tensor independent of coordinates and this causes its ordinary derivative to vanish. This feature is employed in weak field approximations and algebraically degenerate metrics where the Lorentz metric appears as a function of the metric tensor. We may also identify a metric by entering the command ''metric(g)'' (all ITMS function names and definitions are written with double quotes in this text) which enables MACSYMA to raise and lower indices of a tensor with respect to the tensor named g. With such a definition we may employ the ''contract'' command so that the statement ''contract(g([i,j],[])*g([],[j,k]))'' returns ''delta([i],[k])''. The Kronecker delta as well as the generalized Kronecker delta are also used in the contract routine for index substitution. The function ''delta([],[])'' is the dimension of the manifold with a default of 4.

In contrast to hand calculations, one of the difficulties faced with indicial tensor manipulation is the ease with which one may create expressions with more than one covariant and contravariant dummy index with the same symbol. To avoid the error we employ an algorithm in ITMS whereby dummy indices are always represented by the set %1,%2,...%n. Whenever a dummy index is generated, a counter is increased by one and appended onto the % symbol to form a new index. For a given metric the calculation of a curvature tensor may cause the counter to reach a large number. However, expressions with multiple dummy indices are avoided. Clearly, in such a calculation, many of the terms are capable of being combined, differing only in the index number. Simplification of this kind is carried out by expanding the expression and applying the function ''rename'' which resets the counter to zero and renames dummy indices in each of the expanded terms. The resulting expression is then the same order of complexity as one would find by hand calculation.

Multiple covariant differentiation of any tensor density is based upon an algorithm described elsewhere (ref.4). The resultant expression may be expressed in terms of Christoffel symbols or evaluated for a particular indicial metric if one has been defined.

Other features we have implemented include a function called ''show'' which displays any indexed object with its appropriate covariant and contravariant indices. A function called ''nterms'' will tell the user the upper limit to the number of terms an expression would have if fully expanded. This is useful for avoiding the manipulation of an expression which is so large that the system is not capable of simplifying it. If too large the user may use ITMS to simplify the subexpressions and combine them later or decide a new approach to the calculation is appropriate. A function called ''defcon'' allows one to impose various types of contraction properties such as whether a given vector is null or whether a given tensor is trace free. A function ''geodesic'' evaluates expressions in coordinate systems in which undifferentiated Christoffel symbols are set to zero. ITMS has pattern matching routines to enable the user to apply various conditions on differentiated tensors such as the Lorentz conditions. Another feature is the ability of ITMS to perform differentiation with respect to the metric tensor and its derivatives. This enables ITMS to compute field equations for alternative relativistic Lagrangians (ref.5). ITMS also manipulates the numerical tensor densities.

To exemplify the speed and ability of the system we can carry out verification of the Bianchi identity (see any text on relativity) given by $R^{\underset{\vee}{ij}(kl;a)} = 0$ by expanding the Riemann tensor in terms of Christoffel symbols and employing the simplification routines of ITMS in 4 seconds cpu time. Here the parentheses imply symmetrization of enclosed indices, the semicolon is covariant differentiation and the hook denotes anti-symmetric indices. As another example, the Balakram identity (ref.6) which is $R^{\underset{\vee}{ij}}{}_{kl;ij} = 0$ can be verified in 40 seconds cpu time.

Many calculations in gravitation are straightforward with ITMS. The definitions of the Christoffel symbols, curvature tensor, and various geometrical objects are programmed in the system as functions of the metric tensor or other geometrical objects. For example we may define the metric tensor and its inverse by commands in ITMS notation such as

```
COMPONENTS(g([i,j],[]), E([i,j],[])+L*(2*H([i,j],[])-E([i,j],[])*H([],[])))
COMPONENTS(g([],[i,j]), E([],[i,j])-L*(2*H([],[i,j])+E([],[i,j])*H([],[])))
```

for the weak field metric approximation defined by the metric tensor components

$$g_{ij} = E_{ij} + L*(2*H_{ij} - H*E_{ij}) \qquad g^{ij} = E^{ij} - L*(2*H^{ij} - H*E^{ij}) \tag{1}$$

Here $E_{ij}$ is the Lorentz metric, $H_{ij}$ is an arbitrary tensor field, H its trace and L is an infinitesimal expansion parameter(ref.7). In this case it is usual to impose the Lorentz condition $H^{ij}_{\ \ ,j} = 0$. For such a metric we can use ITMS to compute the first order Riemann tensor, Einstein tensor and Weyl tensor in less than 10 seconds cpu time with the implementation of the Lorentz condition. While the full manipulative ability of the ITMS system has not been rigorously tested we have had occasion to compute Einstein tensors with fourth order metrics replacing the right hand side of (1). These calculations involved the manipulation of expressions with more than 1000 terms which were contracted and simplified. Thus the memory space available to ITMS is seen to be quite large.

One of the large calculations used to test ITMS involved the study of the gravitation theories of H. Yilmaz. To third order, Yilmaz' metric is (ref.8)

$$\begin{aligned} g_{ij} = E_{ij} &+ 2*L*(H*E_{ij} - 2*H_{ij}) + 2*L^2*(H^2*E_{ij} - 4*H*H_{ij} + 4*H_{ia}H^a_{\ j}) \\ &+ \frac{8}{3}*L^3*(H^3*E_{ij} - 6*H^2*H_{ij} + 12*H*H_{ia}*H^a_{\ j} - 8*H_a^{\ b}H_{ib}H^a_{\ j}) \end{aligned} \tag{2}$$

where H is the trace of $H_{ij}$ which satisfies the Lorentz condition $H^{ij}_{\ ,j} = 0$. ITMS was used to compute the third order Einstein tensor $G_{ab}$ for (2) and subtract from it the third order tensor d'Alembertian of $H_{ab}$. These calculations with ITMS indicate the theory is valid to first order, but when carried to second order difficulties arise which invalidate the theory to all orders. These results are presented elsewhere (ref.9).

An analysis which is ideally suited to ITMS is the study of various metric gravitational theories by using algebraically special metrics (ref.10) where the metric takes the form

$$g_{ij} = E_{ij} - 2*m*L_i L_j \qquad (3)$$

where m is constant, $E_{ij}$ is the Lorentz metric and $L_i$ is a null vector with respect to both $g_{ij}$ and $E_{ij}$. For the metric (3) one also has a number of differential identities which arise from the differentiation of the identity for null vectors, $L^i L_i = 0$. Implementing these identities we can compute the Ricci tensor for (3) in 30 seconds cpu time and verify the well known expressions for the Einstein vacuum field equations in these coordinates (ref.10). We are now attempting to find algebraically special solutions for the Mansouri-Chang equations (ref.11) in addition to the Kilmister-Yang equations (ref.12) which have been discussed in particular coordinate systems (ref.13).

Conformally flat metrics of the form

$$G_{ij} = P*E_{ij} \qquad (4)$$

where P is a scalar and $E_{ij}$ is the Lorentz metric represent ideal candidates for ITMS since simplifications become extensive. For the metric (4) we have examined the class of Riemannian invariants defined in terms of the generalized

Kronecker delta by

$$L(m) = R^{i_1 i_2}_{j_1 j_2} R^{i_3 i_4}_{j_3 j_4} \ldots R^{i_{2m-1} i_{2m}}_{j_{2m-1} j_{2m}} \delta^{j_1 j_2 \ldots j_{2m}}_{i_1 i_2 \ldots i_{2m}} \qquad (5)$$

These invariants are discussed in quantum gravity as they satisfy the Gauss-Bonnet theorem in 2m dimensional spaces. Using ITMS we have expressed the general term L(m) as an ordinary divergence in conformally flat space-times of 2m dimensions and thereby found alternate expressions for the identities of Horndeski (ref.14).

One of our hopes is that ITMS will also have the ability to carry out needed investigations in differential geometry. Many identities in Riemannian geometry are of great importance in physics and new identities will presumably be discovered when computer systems can take the enormous drudgery out of this particular kind of calculation. The difficulty faced is the construction of an algorithm for the complicated symmetry properties which one encounters. We are presently attempting to construct an appropriate algorithm which will permit tensorial manipulations of this type.

A somewhat primitive feature which ITMS currently possesses is the indicial tensor manipulation of non-symmetric metrics. Given a non-symmetric metric and affinity as in the Einstein-Straus theory (ref.15) we can employ ITMS to compute the various geometrical tensors. However, we have not yet implemented appropriate simplification routines.

While we have stressed the relativistic and differential geometrical aspects of ITMS, the package has been used by others and we believe ITMS, with minor modifications, will find applications in many branches of physics.

# APPENDIX

Below we exhibit the output for the weak field approximation in General Relativity(ref.7). (E11) and (E12) are the covariant and contravariant metric tensors to first order in L. The previous commands (C5)-(C8) define the metric tensor to be G, declare the Lorentz metric E to be constant with respect to ordinary differentiation and specify its inner product. (E16) demonstrates that the contraction of the inner product of G with itself, to first order, is equal to the Kronecker delta as expected. The first order Ricci tensor is displayed by (E20). (E21) is the same tensor after implementation of the Lorentz condition. Contracting the Ricci tensor with the metric we obtain the scalar curvature displayed in (E23). We then construct the contravariant Einstein tensor displayed in (E25). A convenient feature of ITMS is seen in (C26) where the metric is redefined as E to enable us to display the ordinary d'Alembertian in the first term of (E28). Then redefining the metric as G we take the covariant divergence of the Einstein tensor to find it vanishes identically as expected.

```
(C5) DECLARE(E,CONSTANT)$

(C6) DEFCON(E)$

(C7) DEFCON(E,E,DELTA)$

(C8) METRIC(G)$

(C9) COMPONENTS(G([I,J],[]),2*(P([],[])*E([I,J],[])-2*P([I,J],[]))*L
     +E([I,J],[]))$

(C10) COMPONENTS(G([],[I,J]),-2*(P([],[])*E([],[I,J])-2*P([],[I,J]))*L
      +E([],[I,J]))$

(C11) SHOW(G([I,J],[]))$

(E11)                        2 (P E    - 2 P   ) L + E
                                  I J      I J        I J
```

```
(C12) SHOW(G([],[I,J]))$
                           I J         I J      I J
(E12)                     E    - 2 (P E    - 2 P    ) L

(C13) RATVARS(L)$

(C14) RATWEIGHT(L,1)$

(C15) RATWTLVL:1$

(C16) SHOW(CONTRACT(RATEXPAND(G([I,J],[])*G([],[J,K]))))$
                                        K
(E16)                              DELTA
                                        I

(C17) RIEMANN([S,U,N],[N])$

(C18) D17,EVAL$

(C19) RICCI:CONTRACT(RATEXPAND(D18))$

(C20) SHOW(RICCI)$
               %1         %1 %2                     %1 %2
(E20)   - 2 L P      + 2 E     L P         - P     E     L E
               U,%1 S            S U,%1 %2    ,%1 %2        S U

                                                           %1
                                                     - 2 L P
                                                           S,%1 U

(C21) SHOW(LORENTZ(RICCI))$
                     %1 %2                     %1 %2
(E21)             2 E     L P        - P      E     L E
                            S U,%1 %2  ,%1 %2         S U

(C22) SC:CONTRACT(RATEXPAND(RICCI*G([],[S,U]))))$

(C23) SHOW(SC)$
                            %1 %2               %1 %2
(E23)                  - 4 P      L - 2 P      E     L
                            ,%1 %2       ,%1 %2

(C24) EINSTEIN:CONTRACT(RATEXPAND((RICCI - SC*G([S,U],[])/2)*G([],[I,S])*
      G([],[J,U])))$
```

```
(C25) SHOW(EINSTEIN)$
             %1 %2  I J           %1 %2   I J        %1 I    %2 J
(E25)     2 E      P       L + 2 P       E    L - 2 P       E     L
                    ,%1 %2        ,%1 %2             ,%1 %2

                                                            %1 J    %2 I
                                                       - 2 P       E     L
                                                            ,%1 %2

(C26) METRIC(E)$

(C27) EINSTEIN:MAKEBOX(EINSTEIN)$

(C28) SHOW(EINSTEIN)$
               I J        %1 %2   I J        %1 I    %2 J        %1 J    %2 I
(E28)     2 []P    L + 2 P       E    L - 2 P       E     L - 2 P       E     L
                          ,%1 %2             ,%1 %2              ,%1 %2

(C29) METRIC(G)$

(C30) COVDIFF(EINSTEIN,J)$

(C31) D30,EVAL$

(C32) CONTRACT(RATEXPAND(D31))$

(C33) SHOW(D32)$

(E33)                                     0
```

## REFERENCES

1. Barton, D.; and Fitch, J. P.: Applications of Algebraic Manipulation Programs in Physics. Rep. Prog. Phys., 35, 1972, pp. 235-314.

2. Pavelle, R.: Unphysical Characteristics of Yang's Pure Space Equations. Phys. Rev. Lett., 37, 1976, pp. 961-964.

3. d'Inverno, R. A.: Algebraic Computing in General Relativity. Gen. Rel. Grav., 6, 1975, pp. 567-593.

4. Pavelle, R.: Multiple Covariant Differentiation- A New Method. Gen. Rel. Grav., 7, 1976, pp. 383-386.

5. Buchdahl, H. A.: The Hamiltonian Derivatives of a Class of Fundamental Invariants. Quart. J. Math., 19, 1948. pp. 150-159.

6. Balakram: A Theorem in Tensor Calculus. J. Lon. Math. Soc., 4, 1929.

7. Weber, J.: General Relativity and Gravitational Waves. Interscience Publ. , 1961, Section 7.1.

8. Yilmaz, H.: New Approach to Relativity and Gravitation. Annals of Physics., 81, 1973, pp. 179-200.

9. Fennelly A. J.; and Pavelle, R.: Nonviability of Yilmaz' Gravitation Theories and his Criticism of Rosen's Theory. Gen. Rel. Grav., 1977, in press.

10. Adler, R.; Bazin, M.; and Schiffer, M.: Introduction to General Relativity. McGraw-Hill Book Company Inc., 1975, Chapter 7.

11. Mansouri, F.; and Chang, L. N.: Gravitation as a Gauge Theory. Phys. Rev., 13, 1976, pp. 3192-3200.

12. Yang, C. N.: Integral Formalism for Gauge Fields. Phys. Rev. Lett., 33, 1974, pp. 445-447.

13. Pavelle, R.: Unphysical Solutions of Yang's Gravitational-Field Equations. Phys. Rev. Lett., 34, 1975, pp. 1114.

14. Horndeski, G. W.: Dimensionally Dependent Divergences. Proc. Camb. Phil. Soc., 72, 1972, pp. 77-82.

15. Schrodinger, E.: Space-Time Structure. Cambridge Univ. Press., 1963, Chapter XII.

## AUTHOR INDEX

Abdali . . . . . . . . . . . . . . 253 3/F10
Ament . . . . . . . . . . . . . . 87 2/A7
Andersen . . . . . . . . . . . . . 161 2/F11
Anderson . . . . . . . . . . . . . 395 5/C8
Avgoustis . . . . . . . . . . . . 21 1/C6
Bogen . . . . . . . . . . . . 11, 75 1/B10
Bulnes . . . . . . . . . . . . . . 461 6/A9
Caviness . . . . . . . . . . . . . 253 3/F10
Char . . . . . . . . . . . . . . . 53 1/E10
Cohen . . . . . . . . . . . . . . 275 4/A9
Cuthill . . . . . . . . . . . . . 131 2/E3
Doohovskoy . . . . . . . . . . . . 473 6/B7
Fateman . . . . . . . . . . . 43, 327 1/D14
Fennelly . . . . . . . . . . . . . 97 2/B3
Geddes . . . . . . . . . . . . . . 405 5/D4
Genesereth . . . . . . . . . 291, 309 4/B11
Golden . . . . . . . . . . . . 1, 109 1/A14
Gosper . . . . . . . . . . . . . . 237 3/E8
Grünbaum . . . . . . . . . . . . . 491 6/C11
Gupta . . . . . . . . . . . . . . 151 2/F1
Ivie . . . . . . . . . . . . . . . 317 4/D9
Karney . . . . . . . . . . . . . . 377 5/B4
Kulp . . . . . . . . . . . . . . . 385 5/B12
Lafferty . . . . . . . . . . . . . 347 4/F11
Lau . . . . . . . . . . . . . . . 395 5/C8
Lewis . . . . . . . . . . . . . . 277 4/A11
Moenck . . . . . . . . . . . . . . 225 3/D10
Moses . . . . . . . . . . . . 123, 275 2/D1
Ng . . . . . . . . . . . . . 151, 177 2/F1
Noor . . . . . . . . . . . . . . . 161 2/F11
Pavelle . . . . . . . . . . . . 75, 97 1/G4
Poole . . . . . . . . . . . . . . 145 2/E9
Pridor . . . . . . . . . . . . . . 253 3/F10
Rothstein . . . . . . . . . . . . 263 3/G6
Spear . . . . . . . . . . . . . . 369 5/A10
Steele . . . . . . . . . . . . 203, 215 3/C2
Stoutemyer . . . . . . . . 315, 425, 447 4/D7
Wahlquist . . . . . . . . . . . . 71 1/F14
Wang . . . . . . . . . . . . . 55, 435 1/E12
Yun . . . . . . . . . . . . . . . 65 1/F8
Zippel . . . . . . . . . . . . . . 361 4/G11

# CONTENTS

FOREWORD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iii 1/A10

PREFACE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . v 1/A12

AUTHOR INDEX . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . vii 1/A1

1. MACSYMA'S SYMBOLIC ORDINARY DIFFERENTIAL EQUATION SOLVER . . . . . . . 1 1/A14
Jeffrey P. Golden

2. A PROGRAM FOR THE SOLUTION OF INTEGRAL EQUATIONS . . . . . . . . . . . 11 1/B10
Richard A. Bogen

3. SYMBOLIC LAPLACE TRANSFORMS OF SPECIAL FUNCTIONS . . . . . . . . . . . 21 1/C6
Yannis Avgoustis

4. AN IMPROVED ALGORITHM FOR THE ISOLATION OF POLYNOMIAL REAL ZEROS . . . 43 1/D14
Richard J. Fateman

5. FLOATING POINT ISOLATION OF ZEROS OF A REAL POLYNOMIAL VIA MACSYMA . . 53 1/E10
Bruce W. Char

6. PRESERVING SPARSENESS IN MULTIVARIATE POLYNOMIAL FACTORIZATION . . . . 55 1/E12
Paul S. Wang

7. ON THE EQUIVALENCE OF POLYNOMIAL GCD AND SQUAREFREE FACTORIZATION PROBLEMS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65 1/F8
David Y. Y. Yun

8. DIFFERENTIAL FORM ANALYSIS USING MACSYMA . . . . . . . . . . . . . . . 71 1/F14
Hugo D. Wahlquist

9. INDICIAL TENSOR MANIPULATION ON MACSYMA . . . . . . . . . . . . . . . . 75 1/G4
Richard A. Bogen and Richard Pavelle

10. PURE FIELD THEORIES AND MACSYMA ALGORITHMS . . . . . . . . . . . . . . 87 2/A7
William S. Ament

11. BLACK HOLES AND RELATIVISTIC GRAVITY THEORIES . . . . . . . . . . . . . 97 2/B3
A. J. Fennelly and Richard Pavelle

12. THE EVALUATION OF ATOMIC VARIABLES IN MACSYMA . . . . . . . . . . . . 109 2/C1
Jeffrey P. Golden

13. THE VARIETY OF VARIABLES IN MATHEMATICAL EXPRESSIONS . . . . . . . . . 123 2/D1
Joel Moses

14. RATIONAL APPROXIMATION TO $e^{-x}$ WITH NEGATIVE REAL POLES . . . . . . . . 131 2/D9
Elizabeth Cuthill

15. TIMING FORMULAS FOR DISSECTION ALGORITHMS ON VECTOR COMPUTERS . . . . 145 2/E9
W. G. Poole, Jr.

16. SYMBOLIC CALCULATIONS IN A FINITE DYNAMIC ELEMENT ANALYSIS . . . . . 151 2/F1
Kajal K. Gupta and Edward W. Ng

17. SYMBOLIC MANIPULATION TECHNIQUES FOR VIBRATION ANALYSIS OF LAMINATED ELLIPTIC PLATES . . . . . . . . . . . . . . . . . . . . . . 161 2/F11
C. M. Andersen and Ahmed K. Noor

18. OBSERVATIONS ON APPROXIMATE INTEGRATIONS . . . . . . . . . . . . . . . 177 2/G13
Edward W. Ng

19. LISP: PROGRAM IS DATA — A HISTORICAL PERSPECTIVE ON MACLISP . . . . 181 3/A8
Jon L White

20. LISP: DATA IS PROGRAM — A TUTORIAL IN LISP . . . . . . . . . . . . . . 191 3/B4
Jon L White

21. DATA REPRESENTATIONS IN PDP-10 MACLISP . . . . . . . . . . . . . . . . 203 3/C2
Guy Lewis Steele Jr.

22. FAST ARITHMETIC IN MACLISP . . . . . . . . . . . . . . . . . . . . . . 215 3/C14
Guy Lewis Steele Jr.

23. ON COMPUTING CLOSED FORMS FOR SUMMATIONS . . . . . . . . . . . . . . . 225 3/D10
Robert Moenck

24. INDEFINITE HYPERGEOMETRIC SUMS IN MACSYMA . . . . . . . . . . . . . . 237 3/E8
R. Wm. Gosper, Jr.

25. MODULAR POLYNOMIAL ARITHMETIC IN PARTIAL FRACTION DECOMPOSITION . . . 253 3/F10
S. K. Abdali, B. F. Caviness, and A. Pridor

26. A NEW ALGORITHM FOR THE INTEGRATION OF EXPONENTIAL AND LOGARITHMIC FUNCTIONS . . . . . . . . . . . . . . . . . . . . . . . . 263 3/G6
Michael Rothstein

27. SUMMATION OF RATIONAL EXPONENTIAL EXPRESSIONS IN CLOSED FORM . . . . 275 4/A9
Joel Moses and Jacques Cohen

28. USER AIDS FOR MACSYMA . . . . . . . . . . . . . . . . . . . . . . . . . 277 4/A11
V. Ellen Lewis

29. THE DIFFICULTIES OF USING MACSYMA AND THE FUNCTION OF USER AIDS . . . 291 4/B11
Michael R. Genesereth

30. AN AUTOMATED CONSULTANT FOR MACSYMA . . . . . . . . . . . . . . . . . . . 309 4/D1
Michael R. Genesereth

31. A MACSYMA COMPUTER-ALGEBRA MOVIE DEMONSTRATION . . . . . . . . . . . 315 4/D7
David R. Stoutemyer

32. SOME MACSYMA PROGRAMS FOR SOLVING DIFFERENCE EQUATIONS . . . . . . . 317 4/D9
John Ivie

33. SOME COMMENTS ON SERIES SOLUTIONS . . . . . . . . . . . . . . . . . . . 327 4/E5
Richard J. Fateman

34. POWER SERIES SOLUTIONS OF ORDINARY DIFFERENTIAL EQUATIONS IN MACSYMA . . . . . . . . . . . . . . . . . . . . . . . . . . . . 347 4/F11
Edward L. Lafferty

35. RADICAL SIMPLIFICATION MADE EASY . . . . . . . . . . . . . . . . . . . 361 4/G11
Richard E. B. Zippel

36. A CONSTRUCTIVE APPROACH TO COMMUTATIVE RING THEORY . . . . . . . . . 369 5/A10
David A. Spear

37. REDUCTION OF THE EQUATION FOR LOWER HYBRID WAVES IN A PLASMA TO A NONLINEAR SCHRÖDINGER EQUATION . . . . . . . . . . . . . . . . . 377 5/B4
Charles F. F. Karney

38. RAY TRAJECTORIES IN A TORUS: AN APPLICATION OF MACSYMA TO A COMPLEX NUMERICAL COMPUTATION . . . . . . . . . . . . . . . . . . . . . . . 385 5/B12
John L. Kulp

39. APPLICATION OF MACSYMA TO FIRST ORDER PERTURBATION THEORY IN CELESTIAL MECHANICS . . . . . . . . . . . . . . . . . . . . . . . . 395 5/C8
John D. Anderson and Eunice L. Lau

40. SYMBOLIC COMPUTATION OF RECURRENCE EQUATIONS FOR THE CHEBYSHEV SERIES SOLUTION OF LINEAR ODE's . . . . . . . . . . . . . . . . . 405 5/D4
K. O. Geddes

41. $\sin(x)^{**}2 + \cos(x)^{**}2 = 1$ . . . . . . . . . . . . . . . . . . . . . 425 5/E10
David R. Stoutemyer

42. MATRIX COMPUTATIONS IN MACSYMA . . . . . . . . . . . . . . . . . . . . 435 5/F6
Paul S. Wang

43. SYMBOLIC COMPUTER VECTOR ANALYSIS . . . . . . . . . . . . . . . . . . . . 447 5/G4
David R. Stoutemyer

44. A NATURAL WAY TO DO SPATIAL LINEAR GEOMETRY IN MACSYMA . . . . . . . 461 6/A9
Juan Bulnes

45. VARIETIES OF OPERATOR MANIPULATION . . . . . . . . . . . . . . . . . . . 473 6/B7
Alexander Doohovskoy

46. PROGRESS REPORT ON THE DETERMINANT OF A RANDOM MATRIX . . . . . . . . 491 6/C11
F. A. Grünbaum

BIBLIOGRAPHY . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 493 6/C13

BLANK PAGE

BLANK PAGE

# PURE FIELD THEORIES AND MACSYMA ALGORITHMS

William S. Ament
Naval Research Laboratory

## SUMMARY

A pure field theory attempts to describe physical phenomena through singularity-free solutions of field equations resulting from an action principle. The physics goes into forming the action principle and interpreting specific results. Algorithms for the intervening mathematical steps are sketched. Vacuum general relativity is a pure field theory, serving as model and providing checks for generalizations. The fields of general relativity are the 10 components of a symmetric Riemannian metric tensor $g_{ij}$; those of the Einstein-Straus generalization are the 16 components of a nonsymmetric $g_{ij}$. Algebraic properties of $g_{ij}$ are exploited in top-level MACSYMA commands toward performing some of the algorithms of that generalization. The light-cone for the theory as left by Einstein and Straus is found and simplifications of that theory are discussed. Attention is called to the need for spinor theories; the algebra of $g_{ij}$ may help in their construction.

## PURE FIELD THEORY (PFT)

A pure field theory (PFT) (ref. 1, final pages) attempts to describe physical phenomena in terms of singularity-free solutions of a set of field equations, the Euler-Lagrange equations of an action principle. The physical wisdom goes into assembling the action integral and into interpreting any specific results; the intervening mathematics appears strictly algorithmic and therefore doable with, and perhaps only with, computer symbol manipulations such as done by MACSYMA. Einstein's general relativity (GR) is a prototype PFT. GR serves both as the physical basis for test algorithms and as model for the following outline of 'formal' PFT.

One has a coordinate manifold of (presumably) four dimensions, parameterized by Gaussian coordinates $x^i$, $i = 1,2,3,4$. Dependent 'fields' having N scalar components $f = f(x^i)$ are assembled, together with their low-order coordinate derivatives $f_{,i}$ , $f_{,ij}$ , ..., into a scalar density L serving as integrand of the action principle LL. The scalar fields f of GR are the 10 components of a symmetric Riemannian metric tensor $\hat{g}_{ij} = \hat{g}_{ji}$.

### Algorithmic Process No. 1 (AP1): Coordinate Independence

Taking the integration of LL over a coordinate region V having smooth boundary B, check that the value of LL is properly invariant to coordinate transformations interior to V.

### AP2: Get the Field Equations as Euler-Lagrange Equations of LL

This amounts to replacing f with f + df, $f_{,i}$ with $f_{,i} + df_{,i}$ etc., throughout L, retaining terms of first degree in df, $df_{,i}$,... in the expansion of the result, and integrating by parts to eliminate, in V, derivatives $df_{,i}$ , $df_{,ij}$ , ... of the 'variations' df. The coefficients of the N df, set to zero, are then the N scalar field equations in the N scalar fields f.

### AP3: Gauge Conditions (Ref. 2)

When the dependent scalars f are components of a tensor such as the $\hat{g}_{ij}$ of GR, then coordinate transformations in V such as T: $x^i \rightarrow x^i + y^i(x^j)$ require corresponding transformations for the indexed field components. For example,

$$d\hat{g}_{ij} \equiv -\hat{g}_{ij,n}\, y^n - \hat{g}_{nj}\, y^n_{,i} - \hat{g}_{in}\, y^n_{,j}$$

is a 'variation' of $\hat{g}_{ij}$ arising from a mere infinitesimal coordinate transformation T. The 10 Euler-Lagrange equations of GR are linear in second derivatives of $\hat{g}_{ij}$ but there are four scalar Bianchi identities of third differential order arising from invariance of LL to the four $d\hat{g}_{ij}$ possible with a four-parameter gauge transformation $y^i(x^j)$. The (unassembled) algorithms for finding the 'gauge variations' and corresponding Bianchi-like identities should be some mix of those of AP1 and AP2.

### AP4: Small Amplitude High Frequency Waves and the Light Cone

If a PFT is to describe physical vacuum somewhere and is to be singularity-free, then the PFT describes vacuum everywhere. The accepted physical vacuum permits gravitational, electromagnetic, and neutrino waves propagating according to a single light-cone or dispersion relation. To find the light cone: In each of the N field equations, substitute f + df*exp($Kb_i x^i$) (K a frequency parameter, $b_i$ a propagation vector, df an infinitesimal scalar amplitude) for each f in each field equation. Expand and retain only terms linear in the df of highest degree in K--which then factors out, along with exp(). The result is N equations each linear and homogeneous in the N amplitudes df, each homogeneous in the $b_i$. Factor the coefficient determinant, finding a sufficient number of quadratic factors $b_i \hat{g}^{ij} b_j$ ≡ bgb to feel sure that bgb = 0 is the light-cone equation. [If no such bgb factor is found or believed, then use what you may have learned for revising L.]

### AP5: GR With Non-Phenomenological Source Terms

The $\hat{g}^{ij}$ of bgb = 0, built from the f and their coordinate derivatives, is necessarily symmetric, and its inverse can be construed as (up to a conformal scalar factor S) the Riemannian metric tensor $\hat{g}_{ij}$ of GR. Use the algorithms of GR to get the Einstein tensor $G_{ij}$, a form in the f, $f_{,i}$,... . Use the field equations for eliminating from $G_{ij}$ the highest derivatives of the f. What's left over is either zero (vacuum) or counts as a $T_{ij}$ energy-tensor source term--again a form in the f and their (low-order) derivatives. [Select, or eliminate the need for selecting, conformal scalar S. Recognize any $T_{ij}$ as implied in L according to Noetherian principles.]

AP6: Neutrinos and Spin One-Half

Unless some of the f in L are spinor variables, there will be no neutrinos among the vacuum waves, or other 'spin-½' structure in the field equations. Thus: prepare a 'spinor version' of L and plod through the foregoing semi-algorithms. [Conversion to 'spinor form' appears algorithmic in GR, starting from a Riemannian metric tensor (ref. 3), but may not be so in other PFT's.]

## EINSTEIN-STRAUS THEORY

The scalar fields of Einstein-Straus (ES) theory (refs. 1 and 4) are the 16 components of a nonsymmetric tensor $g_{ij}$. This $g_{ij}$ is used in an L and in subsequent development in a way suggested in GR, but the $g_{ij}$ is in no way usable for or equivalent to the symmetric Riemannian 10-component metric tensor $\hat{g}_{ij}$ of GR. The ES field equations are derived from an action principle; no one appears to have asked after the 'vacuum waves' of ES theory, their light cone, or its mathematical connection with GR. So we began with the problem of finding the vacuum waves of the ES field equations--equations given in terms of an affine connection or 'gamma' defined as the solution of a 64x64 linear equation system

$$g_{ij,k} = g_{in}\,\Gamma^{n}_{kj} + g_{nj}\,\Gamma^{n}_{ik} \qquad (1)$$

Let the inverse $g^{ij}$ be defined through $g^{ni}g_{nj} \equiv g^{in}g_{jn} = \delta^{i}_{j}$. This leaves another order for the summation over the 'dummy index' n:
$h^{i}_{j} = g^{in}g_{nj}$, with $g^{ni}g_{jn} \equiv (h^{-1})^{i}_{j}$. Let AA = $h^{i}_{i}$ = trace (h),
CC = $(h^{i}_{j}h^{j}_{i})$ = trace $(h^2)$, BB = $(AA^2 - CC)/2$. Then h = $h^{i}_{j}$ satisfies

$$Q(h) = h^4 - AA*h^3 + BB*h^2 - AA*h + 1 = 0; \qquad (2)$$

$Q(h^{-1}) = 0$ by symmetry. Matrix h has generally four eigenvectors V[n] and eigenvalues v[n]:

$$h^i{}_j V[n]^j = v[n]V[n]^i \quad ; \quad h^i{}_j V_i[n] = v[n]V[n]_j \tag{3}$$

One can normalize so that $V[m]_i V[n]^i = \delta[n,m]$ and (summing over the repeated 'eigenindex' n) $V[n]_i V[n]^j = \delta^j{}_i$. The symmetry of Q(h) implies that if $Q(x) = 0$ then $Q(1/x) = 0$ so that if $v[n]$ is an eigenvalue then so is $1/v[n] = v[n']$, say. Thus, eigenindices [n] (which are not tensor indices) run over say 1,1',2,2' and we introduce op: op[n]:=n', op[n']:=n. With this, and with $u[n]^2 = v[n]$, $u[n]u[n'] = 1$, we have $h^i{}_j = v[n]V[n]^i V[n]_j$ and compatible representations

$$g_{ij} = u[n]V[n']_i V[n]_j \quad , \quad g^{ij} = u[n']V[n']^i V[n]^j \tag{4}$$

Thus, the 16-scalar $g_{ij}$ of ES theory has a natural 18-parameter representation with spinor-like (ref. 3) eigenindexing, and supplies what may be called a built-in vierbein provided by the four directions $V[n]^i$, n = 1,1',2,2'.

The ES field equations being in terms of the gammas, we solved (1) for the gammas using $\Gamma^i{}_{jk} = g^{ni} W_{njk}$ with W represented in the manner of (3), (4) through eigenindices as $W_{ijk} = Z[p,q,r]V[p]_i V[q]_j V[r]_k$ say. By exploiting symmetries, the 64x64 problem (ref. 5) of inverting (1) for the gammas reduces to a 10x10 problem for finding Z[p,q,r]. The straightforward MACSYMA solution, giving terms of up to degree 6 in AA, 5 in BB, is computationally useless (as suspected by Schrödinger, ref. 4, p. 111): formally, there are some 472 terms before replacing three scalar symbols by three $h^i{}_j$ matrices.

The ES field equations, however, entail the gammas in symmetrized or internally contracted forms, so that it was possible to use eigenindexing to set them in terms of the basic fields $g_{ij}$ without resort to the formal inversion of (1). The 16x16 determinant of the homogeneous equation system resulting from AP2 was much too big for the computer but could be made tractable: (1) Resolve the equations and the $b_i$ along vierbein directions, as already done for the gammas by the W → Z above. (2) Then bgb has to be two formally identical terms, one in eigenindices 1,1' the other in 2,2'; replace variables having 2,2' indices with random integers. (3) Any $b_i$ given in eigenindex or vierbein components as (b1,b1',b2,b2') = $\underline{b}$ is orthogonal, for any possible $\hat{g}_{ij}$ 'metric', to $\underline{c}$ = (b1,-b1',0,0) and to $\underline{d}$ = (0,0,b2,-b2') in the sense $b_i \hat{g}^{ij} c_j \equiv bgc \equiv cgb \equiv 0$ and bgd $\equiv$ 0. A final such vector $\underline{e}$ = (b1,b1',-b2,-b2') satisfies cge $\equiv 0 \equiv$ dge; cgd $\equiv$ 0 but bge $\neq$ 0 generally. Take the amplitude-tensor $dg_{ij}$ as a 4x4 quadratic form (exterior product) in the near-orthogonal vector system $\underline{b}$,$\underline{c}$,$\underline{d}$,$\underline{e}$, with 16 unknown coefficients as new 'amplitudes'. The substitution diagonalizes the 16x16 equation system into 6x6 and 10x10 blocks. Both blocks appear degenerate (coefficient determinants vanishing). But eliminating equations of the result one at a time gives a sequence of identical bgb factors in which the structure of the symbols of the 1,1' term is matched by that of the integers of the 2,2' term. The resulting eigenindexed bgb then implies a $\hat{g}^{ij}$ from which the light-cone metric is then, via $Q(h) = 0$

$$\hat{g}_{ij} = S\left[(g_{in}h^n{}_j + g_{jn}h^n{}_i) + BB(g_{ij} + g_{ji})\right] \quad , \tag{5}$$

where S is an undetermined conformal scalar. But: the nature of the waves propagating according to the bgb light-cone equation remains unknown, owing to complexity and, particularly, to failure to eliminate 'gauge transformations' mentioned in AP3. (That failure may also account for the degeneracy of the coefficient determinant.)

In GR the bgb = 0 light-cone equation is known a priori; it is asserted in the metric tensor $\hat{g}_{ij}$. In examining final equations for the nature of the 'vacuum waves' one can take $\hat{g}_{ij}$ as locally diagonal, thus rendering symbolically indexed expressions in compact, inspectable forms. No such diagonalization is seen valid in ES theories, and finding bgb may always have to be done with explicit components. If so, the foregoing sketch of a route to bgb will save much time.

Published variants of ES theory use the gammas and are thereby unnecessarily complicated. In Riemannian geometry the gammas, defined in terms of the metric tensor $\hat{g}_{ij}$, are used for forming tensors from derivatives of further scalar and tensor objects. But ES theory is in terms of the $g_{ij}$ from which the gammas are defined, via equation (1), and there are no further objects. Therefore the gammas are superfluous. The ES equations follow GR by using a Riemann tensor given compactly in terms of the gammas and their first derivatives. The Riemann tensor has two basic definitions, equivalent in GR: The coefficient of tensor $T_d$ in $Ta;b;c - Ta;c;b$ is the Riemann tensor $R^d abc$--but there is no $T_a$ in ES theory for which this function of the Riemann tensor might be needed. Alternatively, the lower-indexed Riemann tensor $R_{ijkl}$ is the non-trivial tensor of lowest degree formable from a 'metric tensor' $g_{ij}$ and its derivatives. Handcrafting gives, with

$$[ijk] \equiv \tfrac{1}{2}(g_{ik,j} + g_{kj,i} - g_{ij,k}) \quad , \tag{6}$$

$$\begin{aligned} R_{iknj} = {} & \tfrac{1}{2}(g_{ij,kn} + g_{kn,ij} - g_{in,kj} - g_{kj,in}) + \\ & + IE^{xy}\,([ijx][kny] - [inx][kjy]) \end{aligned} \tag{7}$$

in which $IE^{ij}$ is the (symmetric) inverse to $g_{(ij)} = (g_{ij} + g_{ji})/2$, and order of the indices is to be respected. (Compare eq. (7) with eq. (30) of ref. 6, p. 153.)

The class of PFT's now under consideration is therefore restricted to those starting from the foregoing tensor $R_{ijkm}$ contracted to a curvature scalar R by some multiplier $M^{ijkm}$ concocted from $g^{ij}$, $h^{i}{}_{n}g^{nj}$,..., and then multiplied by various similarly available Jacobians J to form the scalar density L; these forms are essentially unique in GR, where $M^{ijkm} = \hat{g}^{ik}\hat{g}^{jm}$ and $J = (\det(\hat{g}_{ij}))$. In this general ES theory, each term of L can have a scalar coefficient arbitrarily dependent on scalars AA,BB formed from $g_{ij}$.

Tensor $R_{iknj}$ has the familiar symmetries

$$R_{iknj} = R_{njik} = -R_{kinj} \quad .$$

In forming a 'curvature scalar' $M{*}R = M^{iknj}R_{iknj}$, one may assign the same symmetries to the multiplying tensor M. Equation (2) restricts the occurrence of $g^{ij}$ usable in M to essentially four forms $g^{ij}$, $g^{ji}$, $h^{i}{}_{m}g^{mj}$ and $h^{j}{}_{m}g^{mi}$, generically represented here as $F^{ij}$. In view of the symmetries, M can be given as a 10-parameter form Me of symmetrically arranged products $F^{in}F^{jk}$ plus a 3-parameter form Mo of products $F^{ik}F^{nj}$. In addition, from totally antisymmetrized derivatives $ag(i,j,k) = g_{[ij,k]}$ one can assemble a legitimate two-parameter scalar NN = N(a,b,c,d,e,f)ag(a,b,c)ag(d,e,f); tensor N has two additional parameters. Thus symbolic action integrand L = M*R+NN is a form linear in a total of 15 free scalar parameters. Any 'parameter' is actually some function f(AA,BB) depending on the basic fields $g_{ij}$ via the AA,BB of equation (2).

## CONFORMALLY INVARIANT ES THEORY

The present attempt is to assign the foregoing 15 parameters so that LL is conformally invariant, i.e., its value is unchanged by the substitution $g_{ij} \to g_{ij} + wg_{ij}$, where w is an arbitrary infinitesimal scalar function of coordinates. We choose conformal invariance because no plausible alternatives are visible (suggestions are welcome, particularly those having 'spinor' implications), because physicists have said kind things about such conformal invariance, because the problem of assigning conformal scalar S of AP5 and equation (5) becomes eliminated, and most of all, because the choice appears to give a well posed, doable problem having a possibly unique answer.

The present situation with this problem is best described as fluid. The implication, if any, of 'gauge invariance' is not yet understood in this context. Several unmentioned algebraic simplifications make the problem easier than it appears at first glance; not all such algebraic niceties are incorporated, and the present package of computer commands requires too much thinking at the keyboard.

## APPROPRIATE SYMBOL MANIPULATIONS IN MACSYMA

First described are notational and other conventions, then some general purpose commands and functions.

Let $g_{ij}$ → g(i,j) ; $g^{ij}$ → gg(i,j) ; $g_{ij,k}$ → g1(i,j,k)

$g_{ij,kp}$ → g2(i,j,k,p) ; $\Gamma^{i}{}_{jk}$ → gam(i,j,k), $\Gamma^{i}{}_{jh,p}$ → gam1(i,j,k,p),...

(conventional symbol) → (MACSYMA typein and display symbol).

No attempt at displays in textbook format is made; one has to remember that both indices i,j of gg and the first index of gam and gam1 are upper (U) indices whereas the other indices above are lower (L) indices. Thus $g^{ni}g_{nj}$ → gg(n,i)*g(n,j); repeated index n is a 'dummy' index of summation appearing once as U-index and once as L-index. U-index i and L-index j here are 'free' indices appearing once each.

An indexed expression EE is valid only when each free U- or L-index is represented by the same symbol (letter or atom), and occurs only once, in each term of EE, and when any dummy index symbol appears just once in any term as U-index, once as L-index. A validity-checking TEST(EE) is readily constructed. One builds desired forms by 'contraction' on one or more free indices. For example, s = s(i,j,k) = t(i,j,n)*u(n,k) = t*u, where free U-index n in u, L-index n in t, becomes dummy index n in the contracted tensor product s = t*u. To JOIN t,u as s then entails 1) preserving the final free indices i,j,k and 'contraction' dummy index n while 2) changing dummy indices x of say u so as to differ from these of s. This is done by DECLARE'ing i,j,k,n to be constants while changing any item say x of LISTOFVARS(u), found in the similar list of dummies of s, to some new symbol say xrr = CONCAT(x,rr). But this process should not change other atomic symbols such as the AA,BB of (2)--such symbols are thus initially DECLARED constant.

Of course replacement symbol xrr could be found in t; also t,u and a valid resulting s may contain identical, possibly cancelling, terms disguised by having different symbols for the same dummy variable. Thus one wants a function converting each term of an expression EE to consistent canonical indexing. Command h0x(EE,ILIS) does this term by term: ILIS is a list of free indices declared constant. Internal to h0x, YLIS = [y1,y2,...] is an adequately long list of symbols declared constant, and NAMES is an alphanumerically ordered internal list of these names (such as g,gg,gam1) which occur in the term. Suppose ILIS is [b,x,y,a] and f(i,j,b,p,a) is a factor in the formal term of EE; h0x finds this factor as the one containing b, finds its LISTOFVARS [i,j,p], substitutes y1,y2,y3 for i,j,p throughout the term and reconsiders the result with ILIS = [y1,y2,y3,x,y,a],YLIS = [y4,y5,...]. Or if the initial ILIS were empty and the foregoing factor's name f is first in NAMES then y1,y2,y3,y4,y5 are substituted in order for the LISTOFVARS [i,j,b,p,a] and become the new ILIS. At the close, the constants [y1,y2...] of YLIS are replaced by variables p1,p2,... to avoid conflicts in any iteration of h0x. I believe that h0x converts a valid EE to unique form of minimal length when each term of EE has some dummy-containing name occurring just once so as to appear in NAMES, and the order of indices within each named object is unique. Otherwise h0x(EE,[]) will produce an EE with dummy symbols p1,p2 not necessarily in minimal form. Regrettably, this now calls for ad hoc measures and iterations of h0x, which never increase the number of terms.

The symmetry IE(a,b) = IE(b,a) is invoked automatically by a prior DECLARE(IE,COMMUTATIVE); this imposes the canonical ordering IE(a,b) for either form. Declaring ALF commutative, and C constant, then doing LISTOFVARS (APPLY(ALF,[a,y,C,x,b,x2])) produces the alphanumerically ordered list [a,b,x,x2,y]--sans constant C, of course. ALF may analogously be used to order $g_{ij,yx}$ → g2(i,j,y,x) ≡ g2(i,j,x,y) in the latter form, and used in canonical antisymmetrizing commands.

Perhaps the central problem in simplification of dummy-indexed expressions is seen in an example: Let scalar form F be $IE^{xy}(K_{xy}-K_{yx})$. Tensor $IE^{xy}$ → IE(x,y) has been declared 'commutative' so that IE(y,x) appears alphanumerically reordered as IE(x,y). Thus, though nothing is asserted about tensor K, scalar F as contracted from IE,K above is to vanish--it would if the indices of the second factor of F were canonically reordered as permitted by the symmetry of IE. Our dodge has been: substitute the name AK for K in F, do h0x(F,[]) so that the priority in the order of the new indexing goes to AK, resulting formally in F = AK(p1,p2)*(IE(p1,p2)-IE(p2,p1)), whereupon the declared symmetry of IE produces cancellation in the last factor and one gets the wanted F = 0.

Clearly, what one wants is some simplifier that orders dummy indices, of factors in a monomial, taking full account of declared symmetries of tensor factors in which dummies have already been assigned. The problem is complicated by (a) the variety of possible symmetries and antisymmetries, (b) multiple occurrences of tensor names in the monomial, (c) the present necessity to change dummy eigenindex p' = op[p] in step with p = op[p'], (d) the utility of keeping intact the symbols for free indices.

One plausible way to keep free indices, say i,j,k, of a form f = f(i,j,k,dummies), is to contract f with a 'holding tensor' H = H(i,j,k), process the contracted scalar Hf, and then substitute back i,j,k for the p1,p2,p3 of the final result as indexed with priority set by the name H. But this sometimes results in some terms with the anticipated factor H(p1,p2,p3) while other terms have factors say H(p1,p2,op[p1])--making for unwanted thought and typing.

The sketched algorithms of AP2,AP3,AP4 require different types of differentiations. All can (apparently) be done in a single overall command TENSDIFF(EE,NLIS) by supplying appropriate versions of DIFFLIS, listing forms of derivatives, when TENSDIFF calls on it. NLIS lists names of tensors considered differentiable, all other symbols and functions being considered constants. Example: TENSDIFF(f(i,j,p)*g(i,j), [g]) first sees name g in NLIS, goes to a list GSUBS to find g(a,b):=g%[a,b] evaluates EE as FEE:f(i,j,p)*g%[i,j] does DIFF(FEE) returning f(i,j,p)*DEL(g%[i,j]), replaces the MACSYMA symbol DEL by DDEL, DDEL(array member) being specified in DIFFLIS, e.g., DDEL(g%[a,b]):=gl(a,b,ik). Such indexed forms g%, gg%, gam% as may remain are reconverted to initial forms through the array definitions of list GBACK, reversing GSUBS. Index-renaming as in JOIN prevents dummy indices occurring in DIFFLIS from conflicting with those already in EE. The generic differentiation index "ik" is then to be replaced by some chosen symbol, and before any second differentiation the result should (as with an iterated JOIN operation) be boiled down and converted to relatively harmless indices via h0x.

After all differentiations, one goes immediately to eigenindexed forms as much more compact and perspicuous. The basic substitutions are g(x,y):= y*f(x,y) and gg(x,y):= x*f(x,y)--the tensor indices x,y of g,gg become eigenindices and the freestanding factors x,y are in effect the eigenvalues u of equation (4). Function NUFF then sequentially extracts each factor f(p,q) and in its coefficient replaces q with op[p], op[q] with p. Function CRIMP(EE,NAMES) then renames and reorders, term by term, the eigenindices p together with their 'opposites' p' = op(p) in the general manner of h0x, though with priorities as set by the ordered list NAMES of germane function names. With sufficient application of CRIMP, some minimum of ad hoc substitution, and luck, the named objects are canonically indexed and may be factored out, leaving a polynomial P = P(A,...,pl,pl',...) linear in undetermined parameters A. One must eventually allow for p' = op[p] as implying p' = 1/p--but not too soon, for expression p*p'*Z(other indices) represents a sum over eigenindex p with result 4Z. Function CRIMP leaves indices of objects in NAMES as constants, other freestanding indices, like the above p,p', as variables. Function CFDO does sums over such variables: CFDO applied to p*p' yields 4, applied to $p'^{n}*p^{n\pm2}$ yields the scalar AA of equation (2), etc. Polynomial P is reducible to degree 3 in $p^2$ through $Q(p^2) = 0$, equation (2). Requiring P to vanish then gives a set of linear relations among the parameters A, which may now be solved for in familiar ways.

## REMARKS

Described elsewhere in these Proceedings (ref. 7) is a tensor manipulating package ITMS, designed primarily to analyze field equations of GR based on a symmetric metric tensor $\hat{g}_{ij}$. Our developing package is aimed at finding $\hat{g}_{ij}$ as upshot of field equations derived from action integrals based on non-symmetric tensors. There appears to be no significant duplication of ITMS items. I welcome appropriate extensions of ITMS and recommend its use in case of overlapping capabilities.

I call attention to the problem of providing a spinor representation natural for the non-symmetric $g_{ik}$. The present n,n' eigenindexing is suggestive of two-component spinor notation, and the eigenvectors may provide a natural framework for a spinorization.

# REFERENCES

1. Einstein, A.: Meaning of Relativity. Fourth or Fifth ed. Princeton Univ. Press, 1955.

2. Misner, C. W.; Thorne, K. S.; Wheeler, J. A.: Gravitation. W. H. Freeman and Co., 1973.

3. Pirani, F. A. E.: Introduction to Gravitational Radiation Theory. Brandeis Summer Institute in Theoretical Physics 166, 1964.

4. Schrödinger, E.: Space-Time Structure. Cambridge Univ. Press, 1950.

5. Einstein, A.; Kaufman, B.: Algebraic Properties of the Field in the Relativistic Theory of the Asymmetric Field. Ann. of Math., vol. 59, no. 2, 1954, pp. 230-244.

6. McConnell, A. J.: Applications of Tensor Analysis. Dover Publ., 1957.

7. Bogen, R. A.; Pavelle, R.: Indicial Tensor Manipulation on MACSYMA. 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper no. 9 of this compilation.)

# BLACK HOLES AND RELATIVISTIC GRAVITY THEORIES

A. J. Fennelly
Physics and Astronomy Department
Western Kentucky University
Bowling Green, Kentucky 42101

Richard Pavelle
Logicon, Inc.
Hartwell Avenue
Lexington, Massachusetts 02173

## ABSTRACT

We consider all presently known relativistic gravitation theories which have a Riemannian background geometry and possess exact static, spherically symmetric solutions which are asymptotically flat. We show each theory predicts the existence of trapped surfaces (black holes). For a general static isotropic metric we use MACSYMA to compute the Newman-Penrose equations, the black hole radius, the impact parameter and capture radius for photon accretion, and verify asymptotic flatness. These results are then applied to several of the better known gravitation theories. It appears the claims of Hawking, Lightman, Lee and Rosen regarding the existence of black holes in several theories are not valid, and black holes are a natural consequence of present ideas about gravity.

## INTRODUCTION

The subject of black holes has become very popular in recent years. With dozens of papers appearing in scientific journals each month and popular articles in abundance, the subject of black holes is a true mystery since there is no known method for observing them directly if indeed they exist. Opponents develop theories which they believe eliminate black holes entirely while proponents attempt to show that black holes are legitimate or that their existence is temporary in the evolution of certain classes of stars. Our purpose in this paper is to show that black holes are a natural consequence of the basic format of gravitation theories (at this time) when solutions of field equations can be found in exact form and where the background geometry of the space-time is Riemannian. The calculations involved in the analysis are extremely complicated and we would not have attempted this particular problem without the aid of MACSYMA. MACSYMA possesses a number of special purpose relativistic programs as part of the component tensor manipulation system, CTMS, in addition to ITMS (ref.1). Given the metric components as implicit or explicit functions of the coordinates, CTMS can compute all geometrical objects such as Riemann tensors,etc. It also has the capabilities for finding the Newman-Penrose spin coefficients as well as a host of other objects owing to the generality of MACSYMA and CTMS.

## TRAPPED SURFACES AND PHOTON CAPTURE

The line element for a static spherically symmetric metric may be written in isotropic form as

$$dS^2 = e^{2\psi}(dR^2 + R^2 d\Omega^2) - e^{2\phi} dt^2 \tag{1}$$

where $\psi(R)$ and $\phi(R)$. We use isotropic form rather than Schwarzschild coordinates for a glance at the literature shows that (1) with its high degree of symmetry lends itself to closed form solutions more readily than other metrics. For example a closed form solution of the Brans-Dicke theory in Schwarzschild coordinates has never been exhibited (ref.2).

A trapped surface (the physical measure of the radius at which physical laws change) is one for which all geodesic congruences converge, i.e., strike a singularity (ref.3, ref.4). The measure of the convergence of a geodesic is the spin coefficient (ref.5)

$$\rho = l_{\mu;\nu} m^{\mu} \bar{m}^{\nu} \tag{2}$$

where $l_{\mu}$ is the tangent vector to an outward directed null geodesic congruence, the semi-colon is covariant differentiation and $m^{\mu}$ is the complex vector spanning the celestial sphere. The vectors $l_{\mu}$, $m_{\mu}$ and $\bar{m}_{\mu}$ are combined with an ingoing tangent vector $n_{\mu}$ to form a complex null tetrad. The metric is given by

$$g_{\mu\nu} = l_{(\mu} n_{\nu)} - m_{(\mu} \bar{m}_{\nu)} \tag{3}$$

where ( ) is symmetrization. The tetrad obeys usual inner product rules (ref.5).

The isotropic metric (1) may be written in terms of a new luminosity coordinate by the transformation

$$e^{\phi} dt = e^{\phi} dv + e^{\psi} dR \tag{4}$$

which gives the transformed metric (1) as

$$ds^2 = -e^{2\phi} dv^2 - 2e^{\phi+\psi} dv dR + R^2 e^{2\psi} d\Omega^2 \tag{5}$$

The null tetrad components are easily found, and the complex expansion of the null congruence is then found by MACSYMA to be

$$\rho \propto 1 + R\psi' \tag{6}$$

where $\psi' = d\psi/dR$. The expansion $\rho$ will be negative and a trapped surface will form only if $1 + R\psi' < 0$. Clearly, a large class of metrics will satisfy this condition for some critical finite value(s) of the radius which we denote by $R_t$.

This trapped surface location is coordinate dependent. For comparison we shall wish to transform the expression $R_t$ to Schwarzschild coordinates by choosing the coordinate system in which we redefine the radius by $r = Re^{\psi}$. Thus having found the trapped location for (1) we easily find $r_t$.

For a metric to represent the gravitational field of an isolated particle it is necessary that the field vanish asymptotically at large distances from the particle and the space-time reduce to that of special relativity. The invariant measure of ''asymptotic flatness'' is satisfied if the Weyl invariant

$$\Psi_2 \equiv -1/2\ C_{abcd}\ l^a n^b (l^c n^d - m^c \bar{m}^d) \qquad (7)$$

vanishes asymptotically as R where $C_{abcd}$ is the Weyl tensor. For the metric (1) we find CTMS gives the following expression for the Weyl invariant as

$$\Psi_2 = \frac{e^{-2\psi}}{12R} \left(\psi' - \phi' + R\left(\phi'' + (\phi')^2 - 2\phi'\psi' - \psi'' + (\psi')^2\right)\right) \qquad (8)$$

It is well known that General Relativity predicts both the existence of a trapped surface and the logically related physical consequence which is an impact parameter for particle capture residing outside the trapped surface (ref.6). This is a non-Newtonian effect and it is therefore of interest to determine whether other relativistic gravity theories also predict such a phenomenon. The only assumption we make is that the geodesic equations which are valid in General Relativity hold in other theories too. This assumption is reasonable since alternatives to the geodesic equations of motion have not been proposed.

For the metric (1) and motion in the equatorial plane the geodesic equations immediately give two constants of the motion h, and K. These follow respectively from $g_{\phi\phi}\, ds/d\phi = 0$ and $g_{tt}\, ds/dt = 0$. Writing $\lambda = K/h$ as the impact parameter one finds orbital equations which may be put in the form

$$\frac{dR}{d\phi} = \pm \frac{R}{\lambda}\left[R^2 e^{2(\psi-\phi)} - \lambda^2 + \frac{E^2\lambda^2 R^2 e^{2\psi}}{h^2}\right]^{1/2} \tag{9}$$

where $E=0$ for a photon and $E=1$ for a material particle.

We proceed now directly to the photon $E=0$, since material particles are more drastically affected and will simply give more extreme physical behavior. Orbits are stable down to a critical radius given by $R=R_c$. We find a general method for computing the value of $R_c$ is given by simultaneously setting $dR/d\phi = 0$ and $d/dR\ (dR/d\phi) = 0$. These equations also give a corresponding critical impact parameter $\lambda_c$. These conditions are found to give $R_c$ from

$$1 + R(\phi' - \psi')_{R=R_c} = 0 \tag{10}$$

and

$$\lambda_c = R_c e^{2[\phi(R_c)-\psi(R_c)]} \tag{11}$$

for the corresponding capture impact parameter.

## VALUES OF THE PHYSICAL PARAMETERS

We now apply MACSYMA to the equations derived above for the study of various gravitation theories. We adopt the following notation for our physical parameters:

$R_t$ = location(s) of trapped surfaces in isotropic coordinates from (6)

$r_t$ = corresponding location(s) in Schwarzschild coordinates by transformation

$R_c$ = location(s) of photon capture radii in isotropic coordinates from (10)

$r_c$ = corresponding location in Schwarzschild coordinates by transformation

$\lambda_c$ = corresponding impact parameter for photon capture(coordinate independent)

In each theory we use MACSYMA to compute and simplify the physical parameters as well as verify the condition of asymptotic flatness. By equating (1) to the actual metric in each theory we can solve for $\phi$ and $\psi$. Then we use MACSYMA to compute (6), (8), (10) and (11) as well as transform the physical parameters to Schwarzschild coordinates.

A) GENERAL RELATIVITY: The isotropic form of the Reissner-Nordstrom metric is

$$dS^2 = (d\Omega^2 R^2 + dR^2)\left(\frac{M-E}{2R}+1\right)^2\left(\frac{M+E}{2R}+1\right)^2 - \frac{\left(1-\frac{M^2-E^2}{4R^2}\right)^2}{\left(\frac{M-E}{2R}+1\right)^2\left(\frac{M+E}{2R}+1\right)^2}\,dt^2 \qquad (12)$$

where E is the charge of the mass M. We find

$$R_t = 1/2\ (M^2-E^2)^{1/2} \qquad r_t = M\pm (M^2-E^2)^{1/2}$$

$$R_c = 1/4\ (M+K) \pm 1/2\sqrt{2}\ (M+K)^{1/2}(3M+K)^{1/2} \quad \text{where } K\equiv (9M^2-8E^2)^{1/2} \quad \text{and } M\leq K\leq 3M \qquad (13)$$

$$r_c = 3M/2 \pm 1/2\ (9M^2-8E^2)^{1/2} \qquad \lambda_c = (3M+K)^{3/2}/(\sqrt{2}\ (M+K)^{1/2})$$

The results for the trapped surface location are known whereas the form of the metric (12) and the photon capture parameters appear to be new. Setting E = 0 in (13) the parameters become

$$R_t = M/2 \qquad r_t = 0{,}2M$$
$$R_c = M(1 \pm \sqrt{3}/2) \qquad r_c = 0{,}3M \qquad \lambda_c = M \cdot 3\sqrt{3} \tag{14}$$

all of which are known (ref.7) and confirm the validity of our computations.

B) Rosen's Theory (ref.8) : This theory has received wide attention recently and is presently the most popular alternative to General Relativity. One of the reasons for this is the belief that the theory does not predict the existence of black holes. We shall now see this claim is false. Rosen's metric is

$$ds^2 = e^{2M/R}(dR^2 + R^2 d\Omega^2) - e^{-2M/R}dt^2 \tag{15}$$

and we find

$$R_t = M \qquad r_t = M \cdot e$$
$$R_c = 2M \qquad r_c = 2M\sqrt{e} \qquad \lambda_c = 2Me \tag{16}$$

We now see trapped surfaces do exist in this theory as well as capture radii and photon impact parameters.

C) Brans-Dicke Theory: We use the metric in its standard form (ref.2) to find

$$R_t = \frac{M\left(\pm \mathrm{SQRT}\left(\frac{3\omega}{2} + 2\right) + \omega + 1\right)}{2\omega + 3}$$

$$r_t = \frac{(R_t - \alpha)^{1-P}\,(R_t + \alpha)^{P+1}}{R_t}$$

$$R_c = M\left(\frac{\mathrm{SQRT}(3\omega + 4)}{\mathrm{SQRT}(4\omega + 6)} \pm 1\right) \tag{17}$$

$$r_c = \frac{(R_c - \alpha)^{1 - P} (R_c + \alpha)^{P + 1}}{R_c}$$

$$\lambda_c = \frac{(R_c - \alpha)^{1 - \frac{2\alpha}{M} - P} (R_c + \alpha)^{P + \frac{2\alpha}{M} + 1}}{R_c}$$

where

$$P \equiv \frac{(\omega+1)\sqrt{2}}{\sqrt{\omega+2}\,\sqrt{2\omega+3}}$$
$$\alpha \equiv \frac{M\sqrt{\omega+2}}{\sqrt{2}\,\sqrt{2\omega+3}} \qquad (18)$$

Here too we find a contradiction with earlier results which claimed that Brans-Dicke black holes are identical to those of General Relativity (ref.9). Note that trapped surfaces do not form unless the coupling constant $\omega$ is negative. Also, (17) reduce to (14) as $\omega$ becomes infinite as one would expect since this is the asymptotic correspondence limit of the Brans-Dicke theory.

D) Yang-Kilmister Theory (ref.10) : Two solutions of the Yang-Kilmister equations are given as (ref.11)

$$ds^2 = (1-M/R)^2 \, (dR^2 + R^2 d\Omega^2 - dt^2) \qquad (19)$$

and

$$ds^2 = (1+M/2R)^4 \, (dR^2 + R^2 d\Omega^2) - dt^2 \qquad (20)$$

which give respectively

$$\rho = R/R-M \tag{21}$$

$$R_t = M/2 \quad r_t = 2M \quad R_c = M/2 \quad r_c = 2M \quad \lambda_c = 2M \tag{22}$$

The first solution (19) is peculiar as it implies, from (21), an impenetrable barrier at $R = M$ corresponding to $r = 0$ in Schwarzschild coordinates. The second solution exhibits more unusual behavior since the trapped surface location, capture radius and impact parameter reside at the same radius in Schwarzschild coordinates. These results are not surprising since it has been shown, using MACSYMA, that these metrics are unphysical (ref.12) by possessing solutions which give incorrect physical predictions.

E) Lightman-Lee Theory (ref.13) : A metric for this theory is

$$ds^2 = \left[\frac{2R-M}{2R-3M}\right]^2 (dR^2 + R^2 d\Omega^2) - \left[\frac{2R-M}{2R+M}\right]^2 dt^2 \tag{23}$$

which yields

$$\begin{aligned} R_t &= M/2(3\pm\sqrt{6}) \qquad r_t = M/2(5\pm2\sqrt{6}) \\ R_c &= M/2(3+2\sqrt{3}) \qquad r_c = M/2(5+3\sqrt{3}) \qquad \lambda_c = M/2(7+4\sqrt{3}) \end{aligned} \tag{24}$$

It has been claimed (ref.14) that (23) does not contain a black hole radius at $M/2$ and $3M/2$ , where the metric components become singular, since there radii cannot be encountered after travelling a finite affine distance. This claim is invalid since, from (24), we find a trapped surface forms at $M/2(3+\sqrt{6})$ which lies beyond $3M/2$. It is clear a black hole forms in this theory too.

# CONCLUSIONS

We have establised that black holes are a normal rather than a pathological feature of viable gravitation theories. This fact is amplified by the new observation that photon capture and photon impact parameters are also normal occurrences in the behavior of the gravitational field of dense bodies. Thus we have disproven the claim that black holes do not exist in Rosen's theory as well as shown that the trapped surface exists and can be approached in the Lightman-Lee theory. In addition we have shown that Brans-Dicke black holes are quite unlike those of General Relativity. We are now using MACSYMA to investigate a recent attempt introducing Quantum theory into the subject of black holes in the study of the ''evaporation of black holes'' in which particles can tunnel out of the trapped surface. These results will be presented elsewhere.

## REFERENCES

1. Bogen, R. A.; and Pavelle, R.: Indicial Tensor Manipulation on MACSYMA. Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper No. 9 of this Compilation).

2. Brans, C.; and Dicke, R. H.: Mach's Principle and a Relativistic Theory of Gravitation. Phys. Rev., 124, 1961, pp. 925-935.

3. Penrose, R.: Gravitational Collapse and Space-Time Singularities. Phys. Rev. Lett., 14, 1965, pp. 57-59.

4. Hawking, S. W.: Occurrence of Singularities in Open Universes. Phys. Rev. Lett., 15, 1965, pp. 689-690.

5. Newman E.; and Penrose, R.: An Approach to Gravitational Radiation by a Method of Spin Coefficients., J. Math. Phys., 3, 1962, pp. 566-578.

6. Darwin, C.: The Gravity Field of a Particle,I. Proc. Roy. Soc. London A., 249, 1958, pp. 180-194.

7. Misner, C. W.; Thorne K. S.; and Wheeler J. A.: Gravitation. W. H. Freeman, and Co., 1973, pp. 921-924.

8. Rosen, N. A Bi-Metric Theory of Gravitation. Gen. Rel. Grav., 4, 1973, pp. 435-448.

9. Hawking, S. W.: Black Holes in the Brans-Dicke Theory of Gravitation. Commun. Math. Phys., 25, 1972, pp. 167-171.

10. Yang, C. N.: Integral Formalism for Gauge Fields. Phys. Rev. Lett., 33, 1974, pp. 445-447.

11. Pavelle, R.: Unphysical Solutions of Yang's Gravitational-Field Equations. Phys. Rev. Lett., 34, 1975, pp. 1114.

12. Pavelle, R.: Unphysical Characteristics of Yang's Pure Space Equations. Phys. Rev. Lett., 37, 1976, pp. 961-964.

13. Lightman A. P.; and Lee, W. L.: New Two-Metric Theory of Gravity with Prior Geometry. Phys. Rev. D., 8, 1973, pp. 3293-3302.

14. Lightman, A. P.; Press, W.; Price R.; and Teukolsky, S.: Problem Book in Relativity and Gravitation, Problem 17.12, Princeton Univ. Press, 1975.

BLANK PAGE

BLANK PAGE

# The Evaluation of Atomic Variables in MACSYMA *

Jeffrey P. Golden
Laboratory for Computer Science
Massachusetts Institute of Technology

## 1. Introduction

In this tutorial paper, we explore the many issues involving the use of atomic variables, of *names*, in MACSYMA. We hope thereby to gain insight into the complexities of *evaluation* which may sometimes cause frustration to the MACSYMA user. Some of the simpler aspects will be glossed over as they are adequately covered in the MACSYMA Reference Manual (ref. 1), and as we may assume that all MACSYMA users are somewhat familiar with them.

## 2. Evaluation-Free Expressions

We begin by looking at "evaluation-free" expressions, in which names stand for themselves.

```
(C1) FACTOR(X^2-Y^2);
(D1)                            - (Y - X) (Y + X)
```

The basic idea in the above example is clear to the MACSYMA user. We wish to factor the polynomial $x^2-y^2$ over the integers, so we type in the command line shown at (C1), obtaining the answer at (D1). X stands for itself and Y stands for itself.

## 3. Implicit Assignment

Now, we decide to expand the result (D1). We may type

```
(C2) EXPAND(D1);
```

or more usually

---

* This work was supported, in part, by the United States Energy Research and Development Administration under Contract Number E(11-1)-3070 and by the National Aeronautics and Space Administration under Grant NSG 1323.

```
(C2) EXPAND(%);
```

obtaining

```
                                           2    2
(D2)                                      X  - Y
```

In this case, we know that D1 or % do not stand for themselves, but rather that they both refer to the expression -(Y-X)(Y+X); D1 because MACSYMA implicitly labelled that expression with "D1", and % because in MACSYMA it refers to the "previous" expression or computation.

## 4. Evaluation

It is important to be clear on the process by which the command lines (C1) and (C2) were handled. These command lines were *evaluated*, meaning that in order to determine the expressions FACTOR and EXPAND were to operate on, their arguments, the expressions X^2-Y^2, D1, or % were *evaluated* (and simplified) first, and this means that the variables or names in them were *evaluated* one time. *Evaluation of names* means that if a name has been implicitly or explicitly assigned a value, that we obtain that value. If a name has not been assigned a value, the evaluator just returns the name itself.

## 5. Explicit Assignment

We know that we can *explicitly* assign a value to a name with the use of : (colon). So, if we wish to hold on to a polynomial, say $x^2$+x+y, and invent a name of our own for it, we can type

```
(C3) POLY1:X^2+X+Y;
                                            2
(D3)                                   Y + X  + X
```

we know that POLY1 and D3 are the same in the sense that they both refer to the same expression, Y+$X^2$+X. We also note that even though X and Y have no assigned values (are "unbound"), and thus evaluation produced no changes in our polynomial, that it has been reordered by the simplifier. Lastly, D3 being an *implicitly assigned name* goes on the LABELS list, while POLY1 being an *explicitly assigned name* goes on the VALUES list, which perhaps is named somewhat confusingly. (These lists have many uses as noted in the manual.) The following should be clear:

```
(C4) POLY1-2*X;
                                            2
(D4)                                   Y + X  - X
```

## 6. MACSYMA Options

We can also use explicit assignment to reset the value of a MACSYMA option. A MACSYMA *option* is simply a name that has been initially assigned a value by MACSYMA, and which directs the performance of MACSYMA a certain way by its current setting. Thus, if we wish to see the computation time elapsed in evaluating command lines, we may type

```
(C5) TIME:TRUE$
time= 1 msec.
(C6) FACTOR(X^3+Y^3);
time= 90 msec.
                                      2          2
(D6)                        (Y + X) (Y  - X Y + X )
(C7) TIME:FALSE$
```

When we reset a MACSYMA option, even if we reset it back to its initial value, it goes on the `MYOPTIONS` list.

```
(C8) [LABELS,VALUES,MYOPTIONS];
(D8) [[C8, D7, C7, D6, C6, D5, C5, D4, C4, D3, C3, D2, C2,

                                             D1, C1], [POLY1], [TIME]]
```

## 7. The EV Command

Often, we only wish to reset the value of a MACSYMA option temporarily, say, for a single computation. We may do this as follows:

```
(C9) SIN(X)*COS(X),EXPONENTIALIZE;
                          %I X     - %I X      %I X     - %I X
                    %I (%E     - %E       ) (%E     + %E       )
(D9)              - --------------------------------------------
                                         4
```

This sets the value of the MACSYMA option `EXPONENTIALIZE`, normally `FALSE`, to `TRUE` only during the evaluation of the expression `sin x cos x`, thus causing the trigonometric expression to be converted to exponential form.

First, let us note that `(C9)` as given above is an easy way for typing in

```
(C9) SIN(X)*COS(X),EXPONENTIALIZE:TRUE;
```

The latter form is also acceptable, but the former abbreviated variant is available for many

MACSYMA options, and may also be introduced by the user by DECLAREing a variable as an EVFLAG (see the manual, p. 120).

We also note that (C9) is an abbreviated syntax for a call to the EV command, and could have been given as

```
(C9) EV(SIN(X)*COS(X),EXPONENTIALIZE);
```

EV is by far the most frequently used command in MACSYMA. The above example on the face of it looks very simple, and indeed, in most instances EV gives the expected result in a straightforward manner. Unfortunately, as we shall see later on in this paper, EV's many variants which lead to its great usefulness, are also the reason for its complexity, in understanding it and in how it is handled by MACSYMA.

## 8. Single Level of Evaluation

Let's now assign to X the value of Z:

```
(C10) X:Z;
(D10)                                  Z
```

We know what typing in $x^2-y^2$ does:

```
(C11) X^2-Y^2;
                                      2     2
(D11)                                Z   - Y
```

Let us now request the value of D2:

```
(C12) D2;
                                      2     2
(D12)                                X   - Y
```

We notice that the value of D2 has not changed even though X has now been assigned a value. This is because MACSYMA ordinarily evaluates expressions (in this case D2) only one time and does not re-evaluate expressions even if doing so would result in further change.

## 9. Multileveled Evaluation

One can request evaluation until no further change takes place by using the INFEVAL ("infinite evaluation") flag of EV, as follows:

```
(C13) D2,INFEVAL;
                                        2    2
(D13)                                  Z  - Y
```

In designing MACSYMA, we chose to ordinarily evaluate expressions only one time as this gives the user much more control over his/her expressions in that he/she can <u>control</u> the number of times evaluation is to take place. In almost every case this is not an important issue as variables appearing in expressions are usually either unbound (stand for themselves) or are bound to expressions containing variables all of which are unbound. Thus, in almost every case, it would make no difference if we evaluated variables only one time or attempted to evaluate them more than once.

However, suppose the user has an expression which is labelled, say, L1, which contains one or more occurrences of the variable A, and that A in turn has been assigned as value a large expression. (One way of accomplishing this easily is by assigning to L1 before assigning to A.) Then, thanks to the evaluation scheme described above, the user can play around with the expression L1, i.e. use L1 in his/her command lines, without fearing that a large expression will be plugged in for A before the user wants this to occur.

(As another example, when the user typed D2; at (C12), the user may have only wanted to see D2 displayed again, rather than wanting additional computation to take place at that point. Or, when the user types VALUES; at MACSYMA, the user wants to see the <u>names</u> of the variables that have been assigned to, rather than their <u>values</u>.)

When the user wants this plug-in to L1 to take place, this may be done simply with MACSYMA by typing any of the following command-lines:

```
EV(L1);        or        L1,RESCAN;        or        L1,INFEVAL;
```

The first two are equivalent, and take advantage of the fact that calling EV causes the expression L1 to be evaluated one extra time, i.e. twice. This is obviously the reason the flag is named "RESCAN". (The reason for this extra evaluation will be gone into further below, when EV is taken up again.)

The above example, however, is actually somewhat artificial. If the user wanted the above effect, it is more usual to either postpone assigning to A until that assignment is needed, or to use the SUBST command when needed to substitute in that large expression for A. However, one circumstance in which a situation similar to that above occurs is when using the SOLVE command, as in the following:

```
(C14) KILL(X)$

(C15) SOLVE(X^3+X+C,X);
```

```
                                      2
                              SQRT(27 C  + 4)
(E15)                         ---------------
                                6 SQRT(3)

                                      C 1/3
(E16)                           (E15 - -)
                                      2
solution

                                                     %I SQRT(3)   1
                                                     ---------- - -
                       %I SQRT(3)   1                    2        2
(E17)         X = (- ---------- - -) E16 - --------------
                           2        2                   3 E16

                                                     %I SQRT(3)   1
                                                   - ---------- - -
                     %I SQRT(3)   1                      2        2
(E18)         X = (---------- - -) E16 - ----------------
                         2        2                     3 E16

                                            1
(E19)                         X = E16 - -----
                                          3 E16
(D19)                         [E17, E18, E19]
```

We note that in order to keep the solutions E17, E18, and E19 to the cubic equation somewhat smaller than they otherwise might be, the label E16 is automatically assigned by SOLVE to a subexpression common to all three solutions. The label E15 is also generated as an auxiliary label. Thus, we gain somewhat in the size of displayed expressions at the expense perhaps of some convenience in manipulating the expressions.

Now, let us look at what might be seen by some as a problem with MACSYMA's evaluation and simplification scheme. Suppose we have

```
(C20) SIN(X)*COS(X);
(D20)                         COS(X) SIN(X)

(C21) EXPONENTIALIZE:TRUE$

(C22) DIFF(D20,X);
```

```
            %I X     - %I X              %I X     - %I X
      %I (%E     - %E      ) SIN(X)   (%E     + %E      ) COS(X)
(D22) ----------------------------- + --------------------------
                    2                             2
```

Note that the EXPONENTIALIZE flag has been reset in the middle of a computation. The result obtained in D22 (which, by the way, is equivalent to COS$^{2}$(X)-SIN$^{2}$(X)) at first sight may be surprising to the MACSYMA user. We see that even though the EXPONENTIALIZE switch has been set to TRUE via C21, that D22 still has SIN's and COS's in it! This can be seen to be a result of MACSYMA's single level evaluation and simplification scheme in its interaction with the rule for differentiation of products. Those parts of the result which are generated by DIFF are scanned and converted into exponentials, whereas the unrescanned subexpressions are unchanged. The user can obviously obtain the probably desired result, i.e. a fully exponentialized expression, by causing a rescan to take place, e.g. by

```
(C23) EV(DIFF(D20,X));
              %I X     - %I X        %I X        - %I X
        %I (%E     + %E      ) (%I %E     + %I %E      )
(D23) - ------------------------------------------------
                               4

                        %I X     - %I X        %I X        - %I X
                  %I (%E     - %E      ) (%I %E     - %I %E      )
                - ------------------------------------------------
                                         4
```

or by

```
(C24) EV(D22);
              %I X     - %I X 2      %I X     - %I X 2
           (%E     + %E      )    (%E     - %E      )
(D24)      -------------------- + --------------------
                    4                      4

(C25) EXPONENTIALIZE:FALSE$
```

(C25 resets EXPONENTIALIZE back to its default value.)

The results D23 and D24 are different for reasons explained in the section on EV below.

The single level evaluation and simplification scheme gives the user the extra flexibility and control desirable in certain circumstances. Also, manipulation of expressions is faster, as expressions are not ordinarily rescanned unless specifically requested by the user. (An exception to this is in MACSYMA's rational function package, where, in order for algorithms to work correctly, it may be necessary for expressions to be consistent with the current environment.) An implementation which automatically rescans expressions whenever flags such as EXPONENTIALIZE are reset since the last time the expressions were scanned is possible, although cumbersome, and it would remove some level of control from the user.

## 10. The EV Command Explained

We have seen several examples of the versatility of the EV command above. The EV command is used to control the environment in which an evaluation and/or simplification are to take place. The general syntax is

```
exp, arg2,...,argn
```

meaning that the expression exp is to be evaluated and simplified in the environment given by the remaining arguments, the argi. For example, noting (C9) above

```
(C9) SIN(X)*COS(X),EXPONENTIALIZE;
```

we see that the intention is that the expression SIN(X)*COS(X) be simplified, i.e. transformed, in the environment where EXPONENTIALIZE is TRUE.

To see how this affects evaluation, we consider the example

```
(C26) X^2+1;
                                        2
(D26)                                  X  + 1

(C27) %,X=3;
(D27)                                   10
```

The expression % (or D26) is to be evaluated in the environment where X has value 3, giving 10. X has value 3 while evaluating % (D26) irrespective of any value X might have in the "outside world". Also, X will revert to its "outside world" (global) value when evaluation of the call to EV in C27 is completed. (By the way, the syntax X:3 may also be used for X=3 here.)

Now, let us see just how the evaluation of the call to EV in C27 takes place. First, the name % is evaluated, giving $X^2+1$, thereby obtaining the expression EV is to work on. In general, names appearing in the first argument to EV are evaluated one time at this stage. Usually, these names are labels which point to (whose values are) the expressions EV is to work on. The evaluation (of the name X) will not take place in a case like EV(X^2+1,X=3); where the name (X) is the left hand side of an equation or assignment. Obviously, the global value of X is not wanted in this case.

Next, X is bound to 3, and the expression $X^2+1$ is evaluated in this environment, giving 10. So, we note that the original expression % was evaluated twice, i.e. one extra time.

Using this information, we can analyze how the command lines `(C23) EV(DIFF(D20,X));` and `(C24) EV(D22);` are handled. In the case of C23, first the values of D20 (which is `COS(X)*SIN(X)`) and of X (which is X) are retrieved. Then, the resulting expression `DIFF(COS(X)*SIN(X),X)` is evaluated, which means, since EXPONENTIALIZE is TRUE and since the evaluation of arguments takes place before DIFF is called, that COS(X) and SIN(X) are converted to exponentials before the differentiation is carried out. Thus, we see that `EV(DIFF(D20,X));` is equivalent here to `DIFF(EV(D20),X);`. In the case of C24, first the value of D22 is retrieved, which is an expression containing both SIN's and COS's and exponentials. Then, this expression is evaluated, which in this case, since EXPONENTIALIZE is TRUE, simply causes the occurrences of SIN(X) and COS(X) to be converted to exponentials.

Noting the above analysis, the examples in the manual following the description of the SUBST command should be clear. There, the differences between *substitution* as performed by the SUBST command and *binding* as performed by EV, as well as the differences in the order in which and extent to which evaluation takes place are illustrated. (The arguments in a call to SUBST are, of course, evaluated before substitution takes place.)

We have seen above how EV may be used to affect evaluation. We have also seen the use of the INFEVAL flag of EV to cause repeated evaluation of an expression until no further change takes place. Now, we will briefly mention other flags of EV which may be used to affect how evaluation and simplification takes place.

Especially when we use EV to plug in solutions obtained by SOLVE, e.g.

```
(C28) X^3+X+C,E19,RATSIMP;
                                          2        2
                                  108 E15   - 27 C   - 4
(D28)                             --------------------
                                     108 E15 - 54 C
```

we may wish one more evaluation than normal to take place, in this case to eliminate the E15. This may be done with the INFEVAL flag of EV, but if we wish to control the number of extra evaluations (usually, only one will be necessary), this may be done with the EVAL flag of EV.

```
(C29) X^3+X+C,E19,EVAL,RATSIMP;
(D29)                                  0
```

In fact, one extra evaluation will take place for each mention of the EVAL flag. EV finds that E19 evaluates to an equation that is used to obtain a value for X. The RATSIMP flag is a so-called EVFUN which is used to obtain the simplification we desire, by composing it around the first argument, i.e. C29 is equivalent to

```
(C29) RATSIMP(X^3+X+1),E19,EVAL;
```

(EXPONENTIALIZE, used above, is called an EVFLAG. It is a true flag, used to affect simplification of trigonometric functions.)

There is also a NUMER flag to EV which is used to obtain numerical, i.e. floating point, answers where possible. E.g.

```
(C30) SIN(1/2)+SQRT(1+%I),RECTFORM,NUMER;
(D30)                    0.45508987 %I + 1.57810968
```

Sometimes, e.g. when the NUMER effect of EV is desired, but the extra evaluation done by EV is not, the NOEVAL flag may be used to indicate that substitutions rather than evaluations are to be used where necessary. (An example of the use of NOEVAL is given later.) EV will also use substitutions rather than binding when the left hand sides of equations in its latter arguments are non-atomic. E.g.

```
(C31) 2*SIN(X)^2+2*COS(X)^2,COS(X)^2=1-SIN(X)^2,EXPAND;
(D31)                                 2
```

EV also plays a role in MACSYMA's noun/verb scheme in converting nouns like 'DIFF ("derivative") into verbs like DIFF ("differentiate"), as noted in the manual.

## II. Program Binding

This section discusses the binding of names to values in function calls and the handling of BLOCK variables. We proceed by considering an example. The following function definition for MYTAYLOR defines a very limited Taylor series capability.

```
(C32) MYTAYLOR(EXPR,VAR,POINT,HIPOWER):=
         BLOCK([RESULT],
               RESULT: SUBST(POINT,VAR,EXPR),
               FOR I:1 THRU HIPOWER
                   DO (EXPR: DIFF(EXPR,VAR)/I,
                       RESULT: RESULT+(VAR-POINT)^I
                                      *SUBST(POINT,VAR,EXPR)),
               RETURN(RESULT))$
```

```
(C33) MYTAYLOR(SIN(X),X,A,3);
                       3                 2
        COS(A) (X - A)    SIN(A) (X - A)
(D33) - --------------- - --------------- + COS(A) (X - A) + SIN(A)
                6                2
```

The definition for MYTAYLOR has four names, EXPR, VAR, POINT, and HIPOWER, which are called the "formal parameters" of the function definition. They are bound in turn to the values of the arguments or "actual parameters" of the function call; in the case of **(C33)**, to **SIN(X)**, **X**, **A**, and 3, respectively, (**X** and **A** are unbound) when the call is handled. When the body (right hand side of the function definition) of **MYTAYLOR** is exited upon completion, these bindings are undone, and **EXPR**, **VAR**, **POINT**, and **HIPOWER** again take on whatever values they may have had prior to the call. We also note that **EXPR** is assigned a new value each time the **DO** statement loops. This, of course, causes no difficulties.

The definition also has a local **BLOCK** variable **RESULT**. Being a **BLOCK** variable, it is treated as unbound upon entering the **BLOCK**, and in this case in the first actual statement of the **BLOCK**, it is assigned to. **RESULT** is reassigned in the body of the **DO** statement, and, noting the last statement of the **BLOCK**, its final value is actually the value returned by the call to **MYTAYLOR**. And, like the formal parameters of the definition, when the **BLOCK** is exited, **RESULT** takes on whatever value it may have had outside the **BLOCK**.

(We note that we can use this last fact to temporarily reassign the value of a MACSYMA option, as in the following example for teaching MACSYMA a possible simplification rule 0^0 → 1. Here, we want simplification turned off while the rule is being set up to avoid getting an error message.

```
(C34) 0^0;
 0
0  has been generated

(C35) BLOCK([SIMP],SIMP:FALSE,TELLSIMP(0^0,1));
rule placed on **
(D35)                        [**RULE1, SIMPEXPT]

(C36) 0^0;
(D36)                                 1             )
```

Lastly, the definition has a local **DO** variable **I**. I is given an initial value of 1 in the definition. This is the value I has the first time through the body of the **DO**. Each successive time through the body of the **DO**, the value of I is incremented by 1. And, just as with **BLOCK** variables, when the **DO** statement is exited, I takes on whatever value it may have had outside the **DO**.

The above example exhibits no real difficulties. When a function call is made, variables are bound to certain values. The values these variables had prior to these bindings are placed on a list, and when the body of the function, **BLOCK**, or **DO** statement is exited, these prior values are retrieved and the variables are reassigned to them.

But, let us exhibit a case that doesn't work so well. Consider

```
(C37) F(X):=SIN(X)+X$
```

```
(C38) F(-X);
(D38)                              - SIN(X) - X
```

This is surely the answer we expected. We note that X was bound to -X during the evaluation of the body of the definition for F. But, what if

```
(C39) F(X):=EV(SIN(X)+X,NUMER)$

(C40) F(1/2);
(D40)                               0.97942555

(C41) F(-X);
(D41)                               SIN(X) + X
```

The intention of the user is to obtain numerical answers in cases like C40. But, notice what happened in evaluating the command line for C41. Variables in EV's first argument are evaluated twice, and X evaluated twice gives -(-X) or X, not the -X the user probably intended.

One way to get around the problem in this case is to use the NOEVAL flag to EV.

```
(C42) F(X):=EV(SIN(X)+X,NUMER,NOEVAL)$

(C43) [F(1/2),F(-X)];
(D43)                    [0.97942555, - SIN(X) - X]
```

Note that SIN is handled by the simplifier, rather than by the evaluator.

In general, however, when EV is used as above in the body of a function definition, a better and sometimes necessary solution is to name one's local program variables (i.e. function, BLOCK, or DO variables) differently from one's symbolic variables (the variables appearing in one's actual expressions). E.g. if one expects that %X will not appear in one's expressions (or in that of a user of one's programs!), then the following will work.

```
(C44) F(%X):=EV(SIN(%X)+%X,NUMER)$

(C45) [F(1/2),F(-X)];
(D45)                    [0.97942555, - SIN(X) - X]
```

Problems like the above occur rarely in using MACSYMA. We are thinking about solutions to it. It is discussed in reference 2 and a possible solution via a change in implementation of MACSYMA is proposed there.

## 12. Single-Quote and Quote-Quote

Single-quote (') and quote-quote ('') are two operators which affect the evaluation of names (and of other forms) in essentially opposite ways. A complete discussion of these operators is given in section 3.2 on Evaluation in the MACSYMA manual, and that discussion will not be repeated here. Essentially, preceding a variable by a single-quote prevents an evaluation from taking place; while preceding a variable by a quote-quote causes an extra evaluation and simplification to take place. The effect of single-quote is at evaluation time, while that of quote-quote is at parse time. Quote-quote is often used to cause re-evaluation of a C-label.

One interesting use of single-quote is when using the INFEVAL flag of EV. Suppose one has an expression named EXPR which one wishes to repeatedly evaluate until no further change takes place. Suppose, however, that EXPR contains a variable, say X, which one would prefer to retain as a name in the expression, even though X is now bound. One simple way of doing this is as follows.

```
EV(EXPR,INFEVAL,X='X);
```

This assigns to X the value of X during the "infinite" evaluation of EXPR, thus causing X to remain unchanged in the process.

(By the way, using single-quote, of course, offers another solution to our problem above, e.g.

```
(C46) F(X):=EV('(SIN(X)+X),NUMER)$

(C47) [F(1/2),F(-X)];
(D47)                   [0.97942555, - SIN(X) - X]
```

)

## 13. Other Issues

To keep this paper reasonably sized, only the evaluation of atomic variables was discussed. Thus, many other evaluation issues were not mentioned. For the sake of completeness, a list of these omitted issues is given here: Other evaluation-forms, e.g. compound statements, the colon-colon (::) operator, LAMBDA notation, APPLY and MAPping, GO and RETURN, predicate evaluation, passing function names into programs and the evaluation of function names, passing array names into programs, the evaluation and simplification of SUM and PRODUCT, the noun-verb scheme, subscripted variables and functions, running interpreted (normal) functions vs. running translated or compiled functions, and debugging what the evaluator has done to you. Many of these issues are discussed at length in the manual, or may be the subject of future papers.

I wish to thank Joel Moses for coaxing me into writing this paper, Ellen Lewis for her helpful assistance, and all members of the Mathlab Group and others at M.I.T. and elsewhere for our many discussions, agreements, and disagreements on the subject of evaluation - a hotly contested issue!

## REFERENCES

1. The Mathlab Group: MACSYMA Reference Manual. Version 8. Lab. Comput. Sci., Massachusetts Inst. Technol., Nov. 1975.

2. Moses, J.: The Variety of Variables in Mathematical Expressions. Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper no. 13 of this compilation.)

# THE VARIETY OF VARIABLES IN MATHEMATICAL EXPRESSIONS

Joel Moses
Laboratory for Computer Science, MIT*

The methods of evaluating mathematical expressions in a symbolic mathematical system differ from system to system. We show that classical computer science evaluation approaches are inadequate for this task. The problem is that one is mixing two worlds - the world of mathematics and the world of programming. An approach which separates these two worlds is indicated, and various alternatives to it are indicated.

Consider the evaluation of the following pair of statements in a programming language such as FORTRAN or PL/I. The statements are written in MACSYMA syntax.

(C1) $y : 1;$
(C2) $x : y + 2;$

After the first statement has been evaluated, the variable $y$ will have the value 1 stored in a cell reserved for $y$. In evaluating the second statement, C2, the value of $y$ is obtained from that cell, a constant 2 is added to it, using integer addition , and the result is stored in the cell reserved for $x$. This process of looking up values in cells temporarily reserved for variables is equivalent to the usual method of evaluation of variables employed in most programming languages.

Now consider a slight variation on the two statements above:

(C1) $x : y + 2;$
(C2) $y : 1;$

Suppose that $y$ has no value at the time the first statement is reached. What is the value to be given to $x$? Different languages will have different results. Some might automatically store some starting value, say 0, for all variables. Others may discover the problem in the compiler and give an error message. In an algebraic manipulation system such as MACSYMA, neither of these actions occurs. The result stored in the cell reserved for $x$ is the expression $y + 2$. This is obtained in the following manner. The identifier $y$ is encountered and the cell reserved for it is examined. This yields the information that $y$ has no value at this time. Thus the result returned for $y$ is the expression $y$ itself. Such an action cannot be taken by an algebraic language which does not have a symbolic expression as a legal data type, and it is one thing which makes algebraic manipulation languages differ from other languages. Next the constant 2 is evaluated as usual. The addition is handled differently. Since we no longer have numbers only, numerical addition becomes

---

*. This work was supported, in part, by ERDA contract Number E(11-1)-3070 and NASA Grant NSG 1323.

simplification of sums. The simplifier may use numerical addition, but in this case cannot, and thus returns the expression $y + 2$ to be stored in the cell for $x$.

Now consider the second statement, C2. The evaluation done here is quite normal, that is, a constant 1 is stored in the cell reserved for $y$. Consider the value for $x$ after this point, however. Either $x$ has the old value of $y + 2$ or else it has the value 3, which utilizes the newly obtained value of $y$. That there is an issue here is due solely to the fact that the variable $x$ has a value involving the symbol $y$. In the usual algebraic language, if $x$ depended on an old <u>numerical</u> value of $y$, and then $y$'s value changed, no one would expect $x$'s value to change automatically.

Let us consider the alternatives for the value of $x$ again. The value $y + 2$ is easy to get, because that is exactly what is stored in the cell reserved for $x$. We claim that users of algebraic manipulation systems want to get the value 3 most of the time. There are several ways of getting that value for $x$. The rest of this paper will discuss such approaches, and the difficulties that they engender.

The basic idea of the alternative approaches is to re-evaluate the value of a variable. Thus in MACSYMA the command:

```
(C3)      x;
```

will return $y + 2$, but

```
(C3')     EV(x);
```

will return 3.

The EV function will, in effect, evaluate the expression $y + 2$ for $x$. Since $y$ now has the value 1, simplifying $1 + 2$ will yield 3. Thus the MACSYMA user can in this case choose either of the alternative values for $x$. The EV command is insufficient in handling more complex cases, however. Furthermore, experience indicates that the value the user would normally want to see is 3, and thus extra work should be required for getting the $y + 2$ rather than the 3, as is now the case.

A simple example, where EV fails to give the desired value, is shown below:

```
(C1")          x : y + 1;
(C2")          y : z + 2;
(C3")          z : w + 3;
```

Consider the possible values for $x$: Using the usual algebraic evaluation scheme, $x$ evaluates to $y + 1$. Using EV($x$), we would get $z + 3$ after simplification. Our user probably wants to see $w + 6$. We could get that by calling EV twice, or EV($x$,EVAL), but that simply exposes the problem with EV, that one may need to hold its hand until one gets the value one desires. The key to getting $w + 6$ automatically is to consider another evaluation strategy; namely a Markovian or infinite evaluation strategy.

The basic idea behind infinite evaluation is to keep evaluating the results until there is no change. The process ends when one obtains a constant or a variable which has not yet been given a value. Such a strategy has recently been introduced into EV with the INFEVAL mode. Thus, EV($x$, INFEVAL) would yield $w + 6$ in the example above.

There are two basic problems with the infinite evaluator strategy. It is not the strategy you want when dealing with usual programming variables. Moreover, when it is clear that you want something like infinite evaluation, it is not precisely infinite evaluation that you want. We shall deal with the latter, and easier, issue first.

Consider a situation which might occur when one uses substitution of variables a number of times in a problem:

$x : \mathrm{f}(y,z);$

$y : \mathrm{g}_1(u,v);$
$z : \mathrm{g}_2(u,v);$

$u : \mathrm{h}_1(r,s);$
$v : \mathrm{h}_2(r,s);$

$r : \mathrm{k}_1(p,q);$
$s : \mathrm{k}_2(p,q);$

What are the possible values for $x$? The usual evaluation strategy will yield f($y,z$). EV($x$) will yield an expression in $u$ and $v$. EV($x$, INFEVAL) will yield an expression in values $p$ and $q$. Suppose you wanted to see $x$ in terms of $r$ and $s$. This request, which is not unreasonable, is hard to satisfy in general using the strategies we have discussed. There is an easy solution, however. This is to make $r$ and $s$ temporarily appear to have no value, and then infinitely evaluate $x$. We call the role that $r$ and $s$ play in this case shadow variables. Shadow variables are variables which have known values, but are temporarily considered to be atomic.

Shadow variables are, in a sense, already in use in MACSYMA in various ways. When solving cubic or quartic equations, certain intermediate results are generated and given E labels. The final result is given in terms of these E labels. The reason for using the E labels is to keep the expression relatively small. We claim that the E labels are acting as shadow variables for those intermediate expressions they possess as values. Unfortunately, there is no easy way to keep the E labels from being evaluated on command. An expression containing them, when evaluated using EV, will substitute the values for the E labels. The shadow variable scheme, when implemented, would allow one to introduce shadow variables and specify exactly when their values are to be shown. There are yet other situations in MACSYMA where a similar need for shadow variables shows up. MACSYMA's constants %E and %PI have numerical values associated with them which are revealed when one evaluates an expression with, say, EV(expression, NUMER). Thus %E and %PI may be said to be shadow variables. Similarly the functions SIN and COS are shadowing their numerical counterparts. Thus EV(SIN(1), NUMER) calls the "value" of the SIN function in order to obtain a numerical result.

In the above we have considered evaluation of mathematical expressions without dealing with the companion operation, that is, simplification. Since these two operations tend to get confused, we would like to indicate a possible distinction. We like to consider evaluation as a relatively straightforward, well-defined, and simple operation whose basic job is to replace variables and functions with arguments by their "values". Simplification, on the other hand is a less well-defined operation which does not usually deal with programming concerns such as variables and their values, but rather with equivalence transformations on the mathematical objects themselves. We would like the result of evaluation to be unique. We know that the results of simplification are often not so well defined and different users will want different results.

It turns out that a classic way to implement simplification algorithms is with a Markov algorithm, i.e., infinite evaluation. Since we indicated that infinite evaluation might be of use in evaluation, it is not surprising that one algebraic manipulation system, SCRATCHPAD, has opted for having only an infinite evaluation scheme. This is reasonable only as long as one avoids writing subroutines and stops using variables in the usual programming sense. In such a case, one can get into unexpected difficulties, with one of the simplest of them shown below:

```
(C1)          f(x) := x + 1;
(C2)          f(x + 2);
```

Consider the value of $f(x + 2)$ called for in C2. In MACSYMA, using the usual evaluation strategy, you would get $x + 3$. But with infinite evaluation for all variables you will get an infinite loop, since the $x$ occurring in the expression $x + 1$ in the definition of $f(x)$ forces one to keep evaluating its value. SCRATCHPAD prevents the user from defining functions in the usual way, but this is clearly unsatisfactory in general.

Infinite evaluation thus has a drawback in that it allows infinite loops. The possibility for looping may be essential when dealing with most Markov algorithms. But mathematicians do not evaluate expressions that way! When $x$ depends on $y$ and $y$ depends on $x$ that leads to a system of equations to be solved and not one to be evaluated or simplified. Evaluation of mathematical expressions requires a finite number of substitutions and no loops are allowed. We shall call "finite evalution" the process which evaluates without bound, but which checks for loops and thus avoids infinite loops. We believe that infinite evaluation has been in vogue in certain symbolic systems due, in part, to a confusion between simplification and evaluation. Simplification algorithms, if implemented as Markov algorithms will, in fact, require loops! If a loop is found in finite evaluation, we shall assume that evaluation stops and an error message is given.

Another approach that has been taken is to recognize that some variables will be evaluated once and others infinitely, and to force the user to choose the mode by a declaration or a change in the spelling of the variable's name. An approach which relies on declarations is essentially the one taken in REDUCE. In addition to our desire for a distinction between finite and infinite evaluation and for a shadow variable capability, we eschew the declaration or the spelling approach because one does not want users of interactive systems to make declarations unless they're absolutely required, as successful interactive systems such as APL and LISP have clearly indicated. In addition, the declaration approach is unnecessarily restrictive, since it does not normally allow a variable to be used in both the usual or finite evaluation modes in the same subroutine, for example.

Hence our goal is to indicate an evaluation strategy that 1) gives the user the usual strategy when he wants it for a given variable, 2) gives him the finite evaluation strategy when it is more appropriate and (3) allows him to switch from one mode to the other while requiring hardly any declarations. This particular feat of magic appears possible when we make the following observations:

1) Variables used inside subroutines are usually intended for programming objectives and not as symbolic data objects. Users of such variables will usually want them to be evaluated just once.

2) Variables used in an interactive step-by-step mode, with the exception of labels, are usually intended as symbolic data objects. Users of such variables will usually desire them to be evaluated finitely. Labels, such as MACSYMA's $C_i$ and $D_i$ labels are not data objects. The values of labels will usually be desired to be evaluated finitely, however.

If we take these observations to heart, then we would evaluate all variables inside subroutines just once, and all variables occurring in step-by-step (top level) calculations finitely. We could allow for exceptions by declaration, but such declarations will rarely be necessary. Yet this doesn't solve the problem. The basic dilemma is that inside a given subroutine one could have the identifier $x$ representing a local variable (which is to be evaluated just once for its value) and implicitly have a data object containing the variable $x$ (which is to be evaluated finitely for its (usually different) value).

Before I describe a proposed solution, let me recall some remarks made to me by the late, famous computer scientist, C. Strachey, in 1965. Strachey said that mathematicians never really understood the concept of a variable. The variables in mathematics are clearly constants. It is computer scientists who were the first to deal with and appreciate variability in mathematical objects.

I was deeply impressed by Strachey's comments and to my sorrow I have learned how misleading they were. Mathematicians, physicists and engineers, I have concluded, have used a much richer concept of variable than computer scientists have ever dreamt of. Since symbolic and algebraic manipulation systems are essentially the only computer systems to attempt to deal with mathemati , in the way it is usually dealt with, they have been most hurt by the interpretation of variables in vogue in computer science. In part, computer scientists have been overly enamored by variability of our variables (e.g., $x : x + 1$), and have only lately learned that there is much to be gained in ease of understanding by restricting variability. In part, and this is a major point of the present effort, variables in computer science have not shown much variety of interpretation. The reason is largely that the data objects in vogue in computer science (i.e., numbers) do not possess much structure.

Getting back to the present subject, we note that one solution is to recognize that there may be several different variables with the same name at the same time throughout a computation. Many languages already allow one to use the same identifier for both a function and a variable, since the usage is so very different. Others might let one use array names which are the same as variable names. Again the usage differentiates them. In mathematics it is common to play such games, some would call them puns, depending on context to give sufficient information regarding the type of the variable intended and its mode of interpretation. In our situation, we claim that there is no acceptable solution unless each variable can essentially have two different values, a regular one and a symbolic one. At any given time, the value chosen is a function of the interpretation assigned to the variable. The remaining questions are largely of how one determines what interpretation to assign.

We are, therefore, led to propose the following evaluation strategy:

Rule 1. A variable used in the top level, step-by-step mode uses its symbolic value, unless a declaration is made to do the contrary. The symbolic value is then evaluated finitely.

Rule 2. A variable used inside a subroutine uses its regular value which is not further evaluated, unless there is a declaration made to do the contrary.

Rule 3. A label used at the top level stores its value in its regular value cell. The value of a label is further evaluated finitely.

Switching modes, an issue we made much of earlier, could be accomplished with EV using the following rule.

Rule 4. In a subroutine, EV of a programming variable first evaluates using the variable's regular value. The result is then evaluated finitely, using only the symbolic values for any variables. Should a variable given to EV not have a regular value or be declared symbolic, its symbolic value (which always exists) is used and evaluated finitely.

We believe that such rules allow for the diversity of usage of variables in symbolic and algebraic manipulation systems that users expect. Since the scheme above has not yet been implemented, we unfortunately do not have practical experience as yet to indicate its acceptance in such a context, but we hope this situation will be remedied soon.

We shall now discuss various approaches which are closely related to the proposal above. The first is that instead of having two value cells for each variable, one would achieve largely the same purpose by automatically renaming one of the variables. For example, any variable occurring inside a subroutine and not declared to be symbolic could be renamed, for example, by automatically attaching the symbol % to the name. Thus, the symbolic and programming variables would be distinct and the values would not clash. The communications between the two modes would be handled by EV still, but slightly differently. For example, suppose we communicate the expression $x+I$ into a subroutine which would like to assign different values to I. Inside that subroutine, we might use the variable J, and then perform SUBSTITUTE (J, 'I, expression). Here, 'I will indicate that we mean the symbolic variable I, rather than the programming variable I.

Another approach, which is closer to what the FORTRAN-based (e.g., FORMAC) rather than the LISP-based systems have attempted is to disallow assignment to symbolic variables and to force users to simulate the Markov algorithm evaluation by explicit substitution. Thus if you wish to substitute 2 for $y$ in an expression, you explicitly make the substitution or similarly indicate it with EV(expression, $y = 2$). This forces the user to separate his mathematical and programming worlds and could avoid some confusions. It does appear to force the user to be more explicit in his evaluations, which may get tiresome. It also necessitates another mechanism for dealing with shadow variables and possibly even with labels for expressions.

## CONCLUSION

This paper discusses various distinctions which can be made regarding evaluation of mathematical expressions: regular evaluation vs. infinite evaluation vs. finite evaluation, regular variables vs. mathematical variables vs. shadow variables vs. labels, simplification vs. evaluation vs. solution of equations. We claim that the unsatisfactory state of evaluation strategies in symbolic systems is due to insufficient use of such distinctions in the past. Yet we can claim to have only begun the discussion about such distinctions and the various mechanisms for implementing them in a human engineered manner.

This paper resulted from discussions that have been going on in the Mathlab Group for the past year. Not surprisingly, a number of positions on evaluation have arisen. We shall mention only two here. In a companion paper, Jeffrey Golden defends MACSYMA's current evaluation strategy. This strategy has changed somewhat in the past year with the introduction of the INFEVAL mode in EV. Another view is held by David Barton. He maintains that mathematicians hardly evaluate expressions. Usually they restrict the range of solutions with side conditions (e.g., let $x^2 = a$ in ...) until only one result is possible. He also maintains that assignment to mathematical variables should appear syntactically different from assignment to programming variables. Substitution also replaces evaluation in many cases in his scheme. The approach of this paper may be viewed as a compromise between such views.

We wish to acknowledge the usefulness of discussions with David Barton and Jeff Golden, as well as with Michael Genesereth, Barry Trager, and Richard Zippel.

BLANK PAGE

BLANK PAGE

# RATIONAL APPROXIMATION TO $e^{-x}$ WITH NEGATIVE REAL POLES

Elizabeth Cuthill
David W. Taylor Naval Ship Research and Development Center

## SUMMARY

This note describes an application of MACSYMA to the generation of an expansion in terms of Laguerre polynomials to obtain approximations to $e^{-x}$ on $[0, \infty)$ of the form

$$\frac{P_m}{(1 + \frac{x}{m})^m}$$

Here $P_m$ is a polynomial of degree m-1 in x. These approximations are compared with those developed by Saff, Schönhage, and Varga [3]. Their's are optimum Chebyshev approximations. In particular, Table 3 contains a comparison ot the maximum errors in the Chebyshev sense showing the superior performance of the approximations in [3] when this norm is used. Table 4 contains a comparison of the least squares errors. In such a comparison, the approximations developed in this paper are superior.

Kaufman and Taylor [4] consider approximations to $e^{-x}$ of the form

$$\frac{P_m}{(1+B_1x)\ (1+B_2x)\ldots(1+B_mx)}$$

where $B_1,\ldots,B_m$ are positive real numbers. In this note we also consider the expansion of $e^{-x}(1+B_1x)\ldots(1+B_mx)$ in terms of Laguerre polynomials. The first few terms of such an expansion are derived with MACSYMA.

## INTRODUCTION

In the few months that we have been working with MACSYMA, we have found that it provides us with a greatly expanded capability for generating and exploring the behavior of a variety of approximations. In this note we discuss one such application of MACSYMA for the generation of rational approximations to $e^{-x}$ on $[0, \infty)$ with negative real poles. There has been considerable interest in the past few years in such approximations because of their importance in developing and analyzing numerical methods for solving certain systems of differential equations [1, 2].

In particular, in a recent paper, Saff, Schönhage, and Varga [3] developed a sequence of rational approximations to $e^{-x}$ for x on $[0, \infty)$ of the form

$$\frac{P_m}{(1 + \frac{x}{m})^m} \qquad m = 1, 2, \ldots \tag{1}$$

(with $P_m$ a polynomial of degree m-1) which are optimum in the Chebyshev norm and converge geometrically to $e^{-x}$ on $[0, \infty)$. On considering this sequence of approximations, a natural question arises - how does it compare to an approximating sequence obtained by using for $P_m$ the first m terms of the expansion of

$$e^{-x} (1 + \frac{x}{m})^m$$

in Laguerre polynomials? Such an expansion can be generated analytically. The availability of MACSYMA allowed us to easily obtain the required expansion to answer some of our questions.

A recent paper of Kaufman and Taylor [4] considers a more general form for the approximating function:

$$\frac{P_m}{(1+B_1x)\ (1+B_2x)\ldots(1+B_mx)} \qquad m = 1, 2, \ldots \tag{2}$$

where again $P_m$ is a polynomial of degree m-1 in x and the $B_m$ are real and positive. They prove an existance theorem for best Chebyshev approximations of this form to $e^{-x}$ on $[0, \infty)$. Their numerical results suggest that the best uniform approximation to $e^{-x}$ from this class has only one pole and for m=2 they prove such a result. Here we consider the first few approximations of this type which can again be generated using the appropriate number of terms of an expansion of $e^{-x}(1+B_1x)(1+B_2x)\ldots(1+B_mx)$ in Laguerre polynomials. In these expansions $P_m$ depends not only on x but on the parameters $B_1,\ldots,B_m$. Nearly optimum values for the $B_i$ in the Chebyshev sense for the first few such approximations are obtained and compared with those obtained in [4].

## RESULTS

The first case considered is the generation of a sequence of approximations to $e^{-x}$ of the form

$$\frac{P_m}{(1 + \frac{x}{m})^m} \tag{3}$$

for m = 1, 2, ..., 10, by a sequence of expansions of the form

$$\sum_{i=0}^{m-1} A_{i,m} L_i(x) \tag{4}$$

where

$$A_{i,m} = \int_0^\infty e^{-x} \left(1 + \frac{x}{m}\right)^m L_i(x)\, e^{-x} dx.$$

do not expect such an approximation to behave well for large m, but for l m we expect it to do reasonably well. Table 1 contains the values of generated by MACYSMA for m = 1, 2, ..., 10. Table 2 contains the eq alent polynomials. The program used for generating such an approximation of order m is given in Figure 1. Figure 2 shows the execution of the program of Figure 1 for m = 4.

Since the Chebyshev approximations are developed in [3] and (4 ) gives a weighted least squares approximation, we expect our maximum absolute error to be larger than that obtained in [3] for an approximation of the same order. This is confirmed by Table 3 which contains estimates of the maximum errors on [0, ∞) for the approximations in Table 2 and in Reference [3]. The relative error for the approximation sequence presented here remains under control somewhat longer than for the minimax approximations of [3]. An estimate of the interval on which the relative error remains under 10% for both sets of approximations is also given in Table 3. Note that beyond that point there will in general be less than one significant figure in the approximation.

Table 4 contains weighted least squares errors in two forms:

$$\left[\int_0^\infty e^{-x} \left(e^{-x} - \frac{P_m}{\left(1 + \frac{x}{m}\right)^m}\right)^2 dx\right]^{1/2} \tag{5}$$

and

$$\left[\int_0^\infty e^{-x} \left(e^{-x}\left(1 + \frac{x}{m}\right)^m - P_m\right)^2 dx\right]^{1/2} \tag{6}$$

Note that for given m and $P_m$, MACSYMA can perform the integration in (6) exactly.

The approximations given by (4) behave somewhat erratically with respect to the error norm (5), but they behave more regularly with respect to the error norm (6) used in generating the approximation.

For the general approximating form in expression (2), the first three approximations to $e^{-x}$ of the parametrized form

$$P_m = \sum_{i=0}^{m-1} A_{i,m}(B_i,\ldots,B_m)L_i(x) \tag{7}$$

where

$$A_{i,m} = \int_0^{\infty} e^{-x}(1+B_1x)\ (1+B_2x)\ldots(1+B_mx)L_i(x)e^{-x}dx \tag{8}$$

were generated. In particular, for m=1,

$$A_{0,1} = \frac{B_1+2}{4} \tag{9}$$

so that the approximating function is

$$\frac{B_1+2}{4(1+B_1x)} \tag{10}$$

The entire set of approximations generated by varying $B_1$ goes through the point x = .5 with a value of .5. Since $|e^{-.5} - .5| = .1065\ldots$, we have a bound on how well (1) can perform in approximating $e^{-x}$ on $[0, \infty)$ for any fixed value of B.

From Table 3 we have that for $B_1 = 1$ in (10), an estimate of the maximum error in approximating $e^{-x}$ on $[0, \infty)$ is .25. This can be improved to .109 by taking $B_1 = 2.435$.

For $m = 2$, we determine

$$A_{0,2} = \frac{B_1B_2+B_1+B_2+2}{4}$$

$$A_{1,2} = -\frac{B_1B_2-2}{8}$$

so that the approximating function has the form

$$\frac{(B_1B_2-2)Z + B_1B_2+2(B_2+B_1) +6}{8(1+B_1Z)\ (1+B_2Z)} \tag{11}$$

As noted in Table 3, when $B_1 = B_2 = .5$, an estimate of the maximum error in using (11) as an approximation to $e^{-x}$ on $[0, \infty)$ is .033. It appears that this can be improved only slightly by changing the values of $B_1$ and $B_2$. Kaufman and Taylor [4] show that the optimum Chebyshev approximation to $e^{-x}$ of the form (2) with negative real poles has $B_1 = B_2$. The optimum pproximation in the Chebyshev sense which they determine has $B_1 = B_2 = .52416$ d has an estimate for the maximum error on $[0, \infty)$ of .02271.

For $m = 3$ we determine

$$A_{0,3} = \frac{3B_1B_2B_3+2(B_1B_2+B_1B_3+B_2B_3)+2(B_1+B_2+B_3)+4}{8}$$

$$A_{13} = -\frac{3B_1B_2B_3+(B_1B_2+B_1B_3+B_2B_3)-2}{8}$$

$$A_{23} = -\frac{3B_1B_2B_3+2(B_1B_2+B_1B_3+B_2B_3)+(B_1+B_2+B_3)-2}{16}$$

With this set of expressions,

$$B_1 = .214 \qquad B_2 = .27 \qquad B_3 = .3$$

or any permutation thereof appears to be near optimum. With this set of parameters our estimate for the maximum error is .019 which compares with the value of .056 from Table 2 for $B_1=B_2=B_3=1/3$ and .00805, the error estimate of Kaufman and Taylor obtained when $B=B_1=B_2=B_3= .27127$ in (2) and $P_3$ was determined to minimize the Chebyshev norm. They determined that this value for B was near optimum.

For convenient reference, a table of the first ten Laguerre polynomials generated by MACSYMA is appended as Figure 3.

## REFERENCES

1. Cody, W. J.; Meindardus, G.; Varga, R. S.: Chebyshev Rational Approximations to $e^{-x}$ in $[0, \infty)$ and Applications to Heat-Conduction Problems. J. Approx. Theory, 2, 1969, pp. 50-65.

2. Cavendish, J. C.; Culham, W. E.; and Varga, R. S.: A Comparison of Crank-Nicolson and Chebyshev Rational Methods for Numerically Solving Linear Parabolic Equations. J. Comp. Physics, 10, 1972, pp. 354-368.

3. Saff, E. B.; Schönhage, A.; and Varga, R. S.: Geometric Convergence to $e^{-x}$ by Rational Functions with Real Poles. Numer. Math. 25, 1976, pp. 307-322.

4. Kaufman, E. H., Jr.; and Taylor, G. D.: Best Rational Approximations with Negative Poles to $e^{-x}$ on $[0, \infty)$. To appear in Pade and Rational Approximations: Theory and Applications (E. B. Saff and R. S. Varga, Eds), Academic Press, Inc.

## TABLE 1

## Coefficients $A_{i,m}$ (See Equation (4))

| m<br>i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | $\frac{3}{4}$ | $\frac{13}{16}$ | $\frac{61}{72}$ | $\frac{891}{1024}$ | $\frac{4433}{5000}$ | $\frac{37289}{41472}$ | $\frac{1711167}{1882384}$ | $\frac{61545067}{67108864}$ | $\frac{35347283}{38263752}$ | $\frac{2974181307}{3200000000}$ |
| 1 | | $\frac{7}{32}$ | $\frac{7}{36}$ | $\frac{359}{2048}$ | $\frac{799}{5000}$ | $\frac{12191}{82944}$ | $\frac{64081}{470596}$ | $\frac{17034691}{134217728}$ | $\frac{4549531}{38263752}$ | $\frac{715994377}{6400000000}$ |
| 2 | | | $\frac{1}{72}$ | $\frac{13}{4096}$ | $-\frac{77}{20000}$ | $-\frac{179}{20736}$ | $-\frac{45061}{3764768}$ | $-\frac{3847297}{268435456}$ | $-\frac{2451601}{153055008}$ | $-\frac{110203799}{6400000000}$ |
| 3 | | | | $-\frac{183}{8192}$ | $-\frac{223}{10000}$ | $-\frac{895}{41472}$ | $-\frac{38721}{1882384}$ | $-\frac{10444889}{536870912}$ | $-\frac{350659}{19131876}$ | $-\frac{220627809}{12800000000}$ |
| 4 | | | | | $-\frac{551}{40000}$ | $-\frac{3919}{331776}$ | $-\frac{76315}{7529536}$ | $-\frac{9363467}{1073741824}$ | $-\frac{4609771}{612220032}$ | $-\frac{334134773}{51200000000}$ |
| 5 | | | | | | $-\frac{2833}{663552}$ | $-\frac{23881}{7529536}$ | $-\frac{5050243}{2147483648}$ | $-\frac{1061711}{612220032}$ | $-\frac{129582367}{102400000000}$ |
| 6 | | | | | | | $-\frac{11313}{30118144}$ | $-\frac{290279}{4294967296}$ | $\frac{310547}{2448880128}$ | $\frac{12658197}{51200000000}$ |
| 7 | | | | | | | | $\frac{3374353}{8589934592}$ | $\frac{487751}{1224440064}$ | $\frac{39102607}{102400000000}$ |
| 8 | | | | | | | | | $\frac{323197}{1224440064}$ | $\frac{90749869}{409600000000}$ |
| 9 | | | | | | | | | | $\frac{71643279}{819200000000}$ |

## TABLE 2

### APPROXIMATING POLYNOMIALS $P_m$ (See Equation (5))

$$P_1 = \frac{3}{4} \qquad P_2 = -\frac{7Z - 33}{32} \qquad P_3 = \frac{Z^2 - 32Z + 152}{144}$$

$$P_4 = \frac{61Z^3 - 523Z^2 - 1878Z + 16814}{16384}$$

$$P_5 = -\frac{551Z^4 - 12384Z^3 + 73632Z^2 + 28896Z - 966216}{960000}$$

$$P_6 = \frac{2833Z^5 - 110015Z^4 + 1480040Z^3 - 7442760Z^2 + 288600Z + 79611960}{79626240}$$

$$\begin{aligned} P_7 = -(&3771Z^6 - 326804Z^5 + 9525750Z^4 - 120883040Z^3 + 621070920Z^2 \\ &- 58785120Z - 7221056400)/7228354560 \end{aligned}$$

$$\begin{aligned} P_8 = -(&3374353Z^7 - 161279391Z^6 + 1981437906Z^5 + 13968908310Z^4 \\ &- 472304862120Z^3 + 3058703597880Z^2 - 190815090480Z \\ &- 43270628481360)/43293270343680 \end{aligned}$$

$$\begin{aligned} P_9 = (&323197Z^8 - 24586616Z^7 + 706666604Z^6 - 9122407392Z^5 \\ &+ 37275042840Z^4 + 284113381440Z^3 - 2860087476960Z^2 + 61728145920Z \\ &+ 49362418082880)/49369423380480 \end{aligned}$$

$$\begin{aligned} P_{10} = -(&23881093Z^9 - 2478867747Z^8 + 104255443296Z^7 \\ &- 2266707280992Z^6 + 26146314869472Z^5 - 126953976270240Z^4 \\ &- 296150820215040Z^3 + 4937695894897920Z^2 + 1012259928960Z \\ &- 9909008224670016 0)/99090432000000000 \end{aligned}$$

TABLE 3.

ERROR ESTIMATES

| m | Maximum Error |  | Interval in which relative error remains less than 10% |  |
|---|---|---|---|---|
|  | For Approximation (3), (4) | From [3] | For Approximation (3), (4) | Using Approximation from [3] |
| 1 | .25 | .16 | - | - |
| 2 | .033 | .025 | [0,2.0] | [0,1.6] |
| 3 | .056 | .015 | [0,2.7] | [0,1.9] |
| 4 | .026 | .0079 | [0,4.4] | [0.2.8],[3.1,4.1] |
| 5 | .0135 | .0031 | [0,4.9] | [0,4.0] |
| 6 | .0080 | .0089 | [0,7.1] | [0,4.7] |
| 7 | .0011 | .00019 | [0,8.2] | [0,7.3] |
| 8 | .0042 | .000121 | [0,8.7] | [0,7.4] |
| 9 | .0059 | .000070 | [0,10.1] | [0,7.4] |
| 10 | .0043 | .000030 | [0,12.5] | [0,8] |

## TABLE 4.

## LEAST SQUARES ERRORS

| m | Estimate of Expression (5) for Approximation of | | Estimate of Expression (6) for Approximation of | |
|---|---|---|---|---|
| | (3),(4) | Reference [3] | (3),(4) | Reference [3] |
| 1 | .136 | .106 | .259 | .274 |
| 2 | .0124 | .0176 | .0432 | .0552 |
| 3 | .0175 | .0105 | .0300 | .0451 |
| 4 | .0082 | .0055 | .0180 | .0328 |
| 5 | .00203 | .00214 | .0060 | .0157 |
| 6 | .000118 | .000620 | .00100 | .0059 |
| 7 | .000267 | .000137 | .00054 | .00158 |
| 8 | .000138 | .000085 | .000352 | .00126 |
| 9 | .0000375 | .000049 | .000125 | .00092 |
| 10 | .00000214 | .0000207 | .0000228 | .000466 |

FIGURE 1. - PROGRAM TO GENERATE $P_m$

```
time:true;
l[n](x,a):=((2*n-1+a-x)/n)*l[n-1](x,a)-((n-1+a)/n)*l[n-2](x,a);
l[0](x,a):=1;
l[1](x,a):=1-x;
fn(x,n):=(1+x/n)**n*exp(-x);
for i:0 thru m-1 do
   (li[i]:ev(l[i](x,0),ratsimp),
    in:-integrate(li[i]*exp(-x)*fn(x,m),x),
   display(a[i,m]:ev(in,x:0,ratsimp)));
displav(p[m]:ev(sum(a[i,m]*li[i],i,0,m-1),ratsimp));
```

FIGURE 2. - EXECUTION OF PROGRAM TO GENERATE $P_4$

```
                 ^K
(C11) m:4;
TIME= 1 MSEC.
(D11)                                    4

(C12) demo(e,1,dsk,elizc);

(C13) TIME:TRUE;
TIME= 1 MSEC.
(D13)                                  TRUE

(C14) L[N](X,A):=((2*N-1+A-X)/N)*L[N-1](X,A)-((N-1+A)/N)*L[N-2](X
,A);
TIME= 1 MSEC.
                        2 N - 1 + A - X
(D14) L (X, A) := --------------- L     (X, A)
       N                  N        N - 1

                                                    N - 1 + A
                                                  - --------- L     (X, A)
                                                        N      N - 2

(C15) L[0](X,A):=1;
TIME= 1 MSEC.
(D15)                             L (X, A) := 1
                                   0

(C16) L[1](X,A):=1-X;
TIME= 1 MSEC.
(D16)                           L (X, A) := 1 - X
                                 1

(C17) FN(X,N):=(1+X/N)**N*EXP(-X);
TIME= 1 MSEC.
                                          X N
(D17)                     FN(X, N) := (1 + -)  EXP(- X)
                                          N

(C18) FOR I:0 THRU M-1 DO
   (LI[I]:EV(L[I](X,0),RATSIMP),
    IN:-INTEGRATE(LI[I]*EXP(-X)*FN(X,M),X),
   DISPLAY(A[I,M]:EV(IN,X:0,RATSIMP)));
                                          891
                                  A     = ----
                                   0, 4   1024

                                          359
                                  A     = ----
                                   1, 4   2048

                                           13
                                  A     = ----
                                   2, 4   4096
                                           183
                                 A     = - ----
                                  3, 4     8192
```

FIGURE 2. - CONTINUED

```
TIME= 9266 MSEC.
(D18)                             DONE

‾(C19) DISPLAY(P[M]:EV(SUM(A[I,M]*LI[I],I,0,M-1),RATSIMP));
                          3        2
                      61 X  - 523 X  - 1878 X + 16814
                 P  = ---------------------------------
                  4                16384

TIME= 40 MSEC.
(D19)                             DONE

_

TIME= 10392 MSEC.
(D20)                       DEMO TERMINATED

(C21)
```

FIGURE 3. - TABLE OF LAGUERRE POLYNOMIALS GENERATED BY MACSYMA
M = 10

```
(C15) L[N](Z,A):=((2*N-1+A-Z)/N)*L[N-1](Z,A)-((N-1+A)/N)*L[N-2](Z,A)$

(C16) L[0](Z,A):=1$

(C17) L[1](Z,A):=1-Z$

(C18) FOR I:0 THRU M-1 DO DISPLAY(LI[I]:EV(L[I](Z,0),RATSIMP))$
                                 LI  = 1
                                   0

                               LI  = 1 - Z
                                 1

                                   2
                                  Z  - 4 Z + 2
                            LI  = ------------
                              2         2

                                3      2
                               Z  - 9 Z  + 18 Z - 6
                       LI  = - --------------------
                         3              6

                           4      3      2
                          Z  - 16 Z  + 72 Z  - 96 Z + 24
                    LI  = ------------------------------
                      4                 24

                      5      4       3       2
                     Z  - 25 Z  + 200 Z  - 600 Z  + 600 Z - 120
             LI  = - ------------------------------------------
               5                        120

                 6      5       4        3        2
                Z  - 36 Z  + 450 Z  - 2400 Z  + 5400 Z  - 4320 Z + 720
          LI  = ------------------------------------------------------
            6                            720

               7      6       5        4         3         2
              Z  - 49 Z  + 882 Z  - 7350 Z  + 29400 Z  - 52920 Z  + 35280 Z - 5040
      LI  = - ----------------------------------------------------------------------
        7                                    5040

             8       7        6         5          4          3          2
      LI  = (Z  - 64 Z  + 1568 Z  - 18816 Z  + 117600 Z  - 376320 Z  + 564480 Z
        8

                                                            - 322560 Z + 40320)/40320

                9       8        7         6          5           4           3
      LI  = - (Z  - 81 Z  + 2592 Z  - 42336 Z  + 381024 Z  - 1905120 Z  + 5080320 Z
        9

                                                 2
                                    - 6531840 Z  + 3265920 Z - 362880)/362880
```

# TIMING FORMULAS FOR DISSECTION ALGORITHMS ON VECTOR COMPUTERS

W. G. Poole, Jr.
College of William and Mary

## SUMMARY

The use of the finite element and finite difference methods often leads to the problem of solving large, sparse, positive definite systems of linear equations. Recently the one-way dissection and nested dissection algorithms have been developed for solving such systems. Concurrently, vector computers (computers with hardware instructions that accept vectors as operands) have been developed for large scientific applications. In reference 1, George, Poole and Voigt analyzed the use of dissection algorithms on vector computers. In that paper, MACSYMA played a major role in the generation of formulas representing the time required for execution of the dissection algorithms. In the present paper the author describes the use of MACSYMA in the generation of those formulas.

## DISSECTION ALGORITHMS

When finite difference or finite element methods are used for approximating solutions of partial differential equations, it is often the case that a large, sparse, positive definite system of linear equations,

$$Ax = b \tag{1}$$

must be solved. We shall assume that the domain over which the differential equation is defined is a square region covered by an $n$ by $n$ grid consisting of $(n-1)^2$ small squares called elements. It follows that $A$ is an $n^2$ by $n^2$ matrix. The ordering of the unknowns at the grid points determines the location of the nonzero components of $A$ and, consequently, the storage and time required to solve the linear system by Gauss elimination.

An ordering of the unknowns called one-way dissection is due to George (see ref. 2). Referring to figure 1, the idea of one-way dissection is first to divide the grid with $m$ horizontal separators. The unknowns in the $m+1$ remaining rectangles are numbered vertically toward a separator and then the

---

This paper was prepared as a result of work supported in part under NASA Contract No. NAS1-14101 at ICASE, NASA Langley Research Center, Hampton, VA 23665 and in part by Office of Naval Research Contract N00014-75-C-0879.

separator nodes are numbered. The problem is to derive formulas for storage and timing requirements and to minimize those formulas with respect to $m$ (see ref. 2).

The second dissection scheme is called nested dissection (again, see ref. 2) and has been shown to be asymptotically optimal (see ref. 3). The idea here is to divide the grid with both horizontal and vertical separators as shown in figure 2. Unknowns in regions 1 - 4 are numbered before those on separators 5 - 7. Each of the regions 1 - 4 is a square and may itself be dissected using horizontal and vertical separators. Thus the idea may be applied recursively and, in the case $n = 2^k-1$, nested dissection will terminate after k-1 steps.

Although both dissection orderings were analyzed in reference 1, only nested dissection will be discussed further here because it is a more important algorithm and the generation of its timing formula was a much more formidable task.

The nested dissection algorithm is nontrivial to describe in detail. It was first developed and analyzed with scalar computers in mind by A. George in the early 1970's. The first attempts at obtaining a timing formula were done by hand and only gave a description of the asymptotic behavior, $0(n^3)$. Later, the first few terms were generated by hand. Then in reference 3, A. George obtained the entire formula with the aid of ALTRAN.

## VECTOR COMPUTERS

The existence of vector computers, i.e., computers with hardware instructions that operate on vectors rather than scalars, raises the question of how effective the dissection techniques are on this rather new class of computers. It is assumed that these computers have basic vector instruction execution times which are of the form

$$T_*(j) = S_* + jP_* , \tag{2}$$

where $T_*(j)$ is the total time for the vector instruction $*$; $S_*$ is an overhead time, called "start-up" time; $P_*$ is the "per-result" time of that instruction; and $j$ is the length of the vector.

The large value of $S_*/P_*$ on currently available vector computers implies that one pays a significant penalty for operation on short vectors; consequently, one would prefer algorithms which permit the longest possible vectors (see ref. 4). However, both of the dissection algorithms work by repeated subdivision of the grid until a minimum operation count is obtained. It is this apparent conflict between the cost of using shorter vectors and the corresponding lower operation counts that was studied in reference 1.

## GENERATION OF FORMULAS

In reference 1, George, Poole and Voigt were interested in obtaining parameterized versions of the timing formulas for the dissection algorithms on vector computers. Such formulas were needed in order to study the effects of varying several parameters. They identified nine parameters characterizing the vector computers: 3 start-up times for vector addition, multiplication, and inner product; 3 per-result times for the same instructions; and 3 scalar operations. Furthermore, there was a parameter, $n$, related to the problem size and another, $\ell$, related to the algorithm which the user could vary at liberty. The goal was to choose $\ell$ so as to minimize the timing formula for a given set of computer parameters and a given problem size. Obtaining the timing formulas was useful in several ways:

(1) With the formulas in hand, one could study the effects of changing values for the parameters. In a hypothetical sense one could try to optimize subject to certain side constraints. In a very practical sense, manufacturers announced changes in the parameter values several times;

(2) There are several options in the implementation of the dissection algorithms. For example, one can use a vector inner product or a vector "outer product" version (see ref. 1). The choice reduces to comparing the time required for a vector inner product versus a vector addition plus a vector multiplication. Timing formulas permitted analysis of such options;

(3) Considerable insight into the vectorization of algorithms was gained. For example, average vector lengths could be studied;

(4) Without the formula, a table of timing values for particular choices of the parameters could be generated by executing a model of the algorithm. However, the coefficients in the formulas could not be generated.

The nested dissection timing formula was generated in the following manner. The execution of the nested dissection algorithm was simulated in a top-down fashion. The top level, level 1, involved several summations of which

$$\sum_{i=1}^{j-1} (2^i - 2)^2 \theta\left(\frac{n - 2^i + 1}{2^i}, \frac{4(n + 1)}{2^i}, 4\right) \tag{3}$$

is typical, where $\theta$ is a procedure at the second level. Each of the second level procedures called several third level procedures, e.g.,

$$\text{THETA(Q,P,K) := CHLSKY(Q) + P LOWSOL(Q) + MODNES(Q,P,K)} \tag{4}$$

CHLSKY, LOWSOL and MODNES are three of the third level procedures defined to be the timing formulas for simple numerical computations, e.g.,

$$\mathrm{CHLSKY}(Q) := \frac{(PA + PM)\ Q^3}{6} + \frac{(SA + SM + PM)\ Q^2}{2} + \left(DSR + \frac{SM}{2} - \frac{SA}{2} - \frac{2\ PM}{3} - \frac{PA}{6}\right) Q - SM \tag{5}$$

is the timing formula for the factorization of a dense linear system. These third level procedures were formulas for factorization, lower solve and upper solve of dense systems and banded systems and matrix modifications of the form

$$A := A - UVW^T \quad . \tag{6}$$

Finally, the bottom level consisted of the parameters which characterize the vector computer. E.g.,

$$SA + Q\ PA \tag{7}$$

is the time for a vector add of length Q.

The second and third levels each consisted of 10 to 15 modules and level 4 consisted of 9 instruction parameters, 1 parameter related to the algorithm and 1 related to the grid size for the problem. The top level module contained several MACSYMA sums of the form

$$\mathrm{SUM}(''(\mathrm{EV}(((2^I-2)^2)*(\mathrm{THETA}((N-2^I+1)/(2^I),4*(N+1)/(2^I),4)), \mathrm{EXPAND})),I,1,J-1) \quad . \tag{8}$$

This is the MACSYMA form of the sum in eq. (3). The entire generated formula consists of over 200 terms and can be found in Appendix B of reference 1. The formula was checked by evaluating it for several sets of parameter values and comparing the results to execution times of a FORTRAN simulation of the algorithm. The one-way dissection formula was generated in a similar, but much more forward, manner.

## CONCLUDING REMARKS

MACSYMA has been shown to be of considerable value in the study of the performance of the nested dissection algorithm when used on hypothetical vector computers. The derived timing formulas lead to an understanding of the effects of varying the parameters which characterize the computers. Options in the algorithm's implementation can be studied as well as the extent to which the algorithm vectorizes.

## REFERENCES

1. George, A.; Poole, W. G., Jr.; and Voigt, R. G.: Analysis of Dissection Algorithms for Vector Computers. ICASE Report No. 76-17, June 8, 1976. (Available as NASA CR-145177.)

2. George, A.: Numerical Experiments Using Dissection Methods to Solve n by n Grid Problems. SIAM J. Numerical Analysis, vol. 14, no. 2, Apr. 1977, pp. 161-179.

3. George, A.: Nested Dissection of a Regular Finite Element Mesh. SIAM J. Numerical Analysis, vol. 10, no. 2, Apr. 1973, pp. 345-363.

4. Lambiotte, J. J., Jr.; and Voigt, R. G.: The Solution of Tridiagonal Linear Systems on the DCD STAR-100 Computer. ACM Trans. Math. Software, vol. 1, no. 4, Dec. 1975, pp. 308-329.

FIGURE 1. - ONE-WAY DISSECTION WITH ORDERING OF UNKNOWNS INDICATED BY NUMBERS (m = 3).

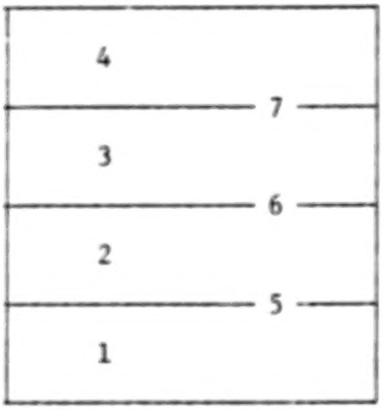
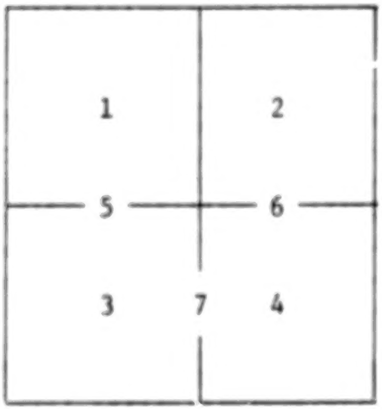


FIGURE 2. - ONE STEP OF NESTED DISSECTION WITH ORDERING OF UNKNOWNS INDICATED BY NUMBERS.



16

# SYMBOLIC CALCULATIONS IN A FINITE DYNAMIC ELEMENT ANALYSIS

Kajal K. Gupta and Edward W. Ng
Jet Propulsion Laboratory

## INTRODUCTION

Since this paper is addressed to an audience primarily interested in symbolic computations, we shall briefly describe the context of engineering mathematics to motivate the computational aspect. The present problem is concerned with prestressed membrane elements with application to the development of large furlable conical spacecraft antennas whose reflector surfaces are made of stretched membranes (Ref. 1). The mathematical aspect involves the application of a finite element method to approximate the membrane deformation as a function of time. The phrase 'dynamic element' is used here to connote time dependent corrections to the static models attacked by the usual finite element method. The general strategy and overall scope of the present application is described by Gupta (Ref. 2) and in the following we shall confine ourselves to the computational problems. Throughout this paper we shall use capital letters for vectors and matrices, and lower case letters for scalars. We shall describe in detail a second order problem for which MACSYMA was used only for checking purpose, and then in brevity a fourth order problem for which a symbolic system is necessary. At the end, some sample output is displayed to indicate the complexity of the computational problem.

## A SECOND ORDER PROBLEM

For the simpler problem we are dealing with a second order time harmonic differential equation in two dimensions, (x,y) and a time variable t:

$$\frac{1}{a^2}\frac{\partial^2 u}{\partial \xi} + \frac{1}{b^2}\frac{\partial^2 u}{\partial \eta^2} = \frac{\rho}{\sigma h}\frac{\partial^2 u}{\partial t^2} \qquad (1)$$

$$\xi = x/a \, , \quad \eta = y/b$$

subject to boundary conditions for the four corners of each rectangular finite element, say, (0,0), (1,0), (1,1) and (0,1):

$$u(0,0) = q_1 \ , \quad u(1,0) = q_2 \ , \ u(1,1) = q_3 \ , \ u(0,1) = q_4$$

Here we are simulating a thin rectangular membrane of thickness h, mass per unit area $\rho$ and uniform tensile force per unit length $\sigma h$, and (a,b) specified the size of the rectangle. The solution is constructed from a second-order expansion of the time harmonic problem, with natural frequency $\omega$, i.e.,

$$u \approx (A_0 + \omega A_1 + \omega^2 A_2)\, Q^T e^{i\omega t} \qquad (2)$$

where $A_0$, $A_1$ and $A_2$ are vector functions of instantaneous nodal displacement and also of the frequency of such motion, and $Q^T$ is a unit vector, all these vectors being dependent only on $\xi$ and $\eta$. We have no formal proof that such expansion converges, but in (Ref. 2) it is given physical arguments and empirical evidence that such expansion does lead to dramatic improvement over the usual finite element approach. Substituting eq. (2) into eq. (1) and equating like powers of $\omega$ render the following equations:

$$\nabla^2 A_0 Q^T = 0 \qquad (3)$$

$$\nabla^2 A_1 Q^T = 0 \qquad (4)$$

$$\nabla^2 A_2 Q^T + \frac{\rho}{\sigma h} A_0 Q^T = 0 \qquad (5)$$

with the corresponding boundary conditions that $A_0 = [q_1, q_2, q_3, q_4]$, $A_1 = 0$ and $A_2 = 0$, where the above symbol [ , , ,] is used throughout the present paper for a row vector, and the superscript T signifies the transpose of a matrix or vector.

At this step we have to choose certain basis functions to form the solutions, for example,

$$A_0 Q^T = c_1 + c_2 \xi + c_3 \eta + c_4 \xi \eta \qquad (6)$$

$$A_1 Q^T = d_1 + d_2 \xi + d_3 \eta + d_4 \xi \eta \qquad (7)$$

$$A_2 Q^T = e_1 + e_2 \xi + e_3 \eta + e_4 \xi \eta + P(c_1, \ldots, c_4, \xi, \eta) \qquad (8)$$

where these coefficients have to satisfy the boundary conditions, and P is a particular integral that satisfied eq. (5). Once a set of basis functions is picked, we need to calculate the A vectors as functions of $\xi$, $\eta$ and the boundary parameters a and b.

The next step concerns the application of the principle of minimum total potential energy. In particular, a sufficient condition for this principle is given by equating the lateral strain energy and the kinetic energy of transverse vibration, i.e.,

$$\Delta U = \Delta T \tag{9}$$

$$\Delta U = \iint_{Area} \frac{\sigma h}{2} \left\{ \frac{1}{a}\left(\frac{\partial u}{\partial \xi}\right)^2 + \frac{1}{b}\left(\frac{\partial u}{\partial \eta}\right)^2 \right\} d\xi\, d\eta \tag{10}$$

$$\Delta T = \iint_{Area} \frac{1}{2} \rho a b \left(\frac{\partial u}{\partial t}\right)^2 d\xi\, d\eta \tag{11}$$

Substitution of eq. (2) into eqs. (10) and (11) gives

$$\Delta U = \frac{1}{2} Q \left(K_{00} + 2\omega^2 K_{02} + \omega^4 K_{22}\right) Q^T \tag{12}$$

$$\Delta T = \frac{1}{2} \omega^2 Q \left(M_{00} + 2\omega^2 M_{02} + \omega^4 M_{22}\right) Q^T \tag{13}$$

where the K's are stiffness matrices and the M's are mass matrices. The zeroth-order terms correspond to the well-known static counterparts in the usual finite element method and the higher-order terms represent dynamic corrections. These matrices are given by

$$K_{ij} = \frac{\sigma h}{2} \iint_{Area} \left( \frac{1}{a} \frac{\partial A_i^T}{\partial \xi} \frac{\partial A_j}{\partial \xi} + \frac{1}{b} \frac{\partial A_i^T}{\partial \eta} \frac{\partial A_j}{\partial \eta} \right) d\xi\, d\eta \tag{14}$$

$$M_{ij} = \frac{\sigma h a b}{2} \iint_{Area} A_i^T A_j \, d\xi\, d\eta \tag{15}$$

The disappearance of the odd terms is due to the symmetry of the problem.

Finally, we can apply the above expressions to eq. (9) and obtain an equation of motion in the form

$$\left[ K_{00} - 2\omega^2 (M_{00} - K_{02}) - \omega^4 (M_{02} - K_{22}) \right] Q = 0 \qquad (16)$$

This is a quadratic eigenproblem and is to be solved numerically. The main use of a symbolic calculator is to prepare and simplify the matrices K's and M's in the form of FORTRAN statements for the inclusion into a numerical program. The symbolic calculation steps may be summarized as follows:

(i) The vectors $[c_1,c_2,c_3,c_4]$ and $[d_1,d_2,d_3,d_4]$ are computed from the boundary conditions.

(ii) $A_0$ and $A_1$ are computed from these two vectors.

(iii) The particular integral P in eq. (8) is chosen. So far the choice has been made from an ad hoc procedure. In the next section we shall describe an attempt towards a more systematic approach for this step.

(iv) From the boundary conditions the vector $[e_1,e_2,e_3,e_4]$ can be calculated in terms of $[c_1,c_2,c_3,c_4]$ which in turn gives the vector $A_2$.

(v) Once the A's are determined, we need to compute the matrices $K_{ij}$ and $M_{ij}$ through symbolic differentiations and integrations.

(vi) The output has to be simplified and formatted for inclusion in a FORTRAN program.

## A FOURTH ORDER PROBLEM

For a plate bending problem we are dealing with the biharmonic equation

$$\frac{1}{a^4}\frac{\partial^4 u}{\partial \xi^4} + \frac{2}{a^2 b^2}\frac{\partial^4 u}{\partial^2 \xi \partial^2 \eta} + \frac{1}{b^4}\frac{\partial^4 u}{\partial \eta^4} = -\frac{1}{b^4}\frac{\partial^2 u}{\partial t^2} \qquad (17)$$

Conceptually the approach is exactly the same as the above problem. The difference in size, however, is two orders of magnitude. There are now 12 boundary conditions, 4 each in u, $\frac{\partial u}{\partial \xi}$ and $\frac{\partial u}{\partial \eta}$. Thus all the vectors and matrices described above are now of dimensions 12 and 12, respectively. The algebraic manipulation is most extensive, and a symbolic system is an absolute necessity here.

For this problem the six steps above basically carry through, with the exception of (iii) which can be somewhat hazardous. We shall illustrate this process in some detail here. Following the same procedure from eq. (1) to eq. (5), we get, with $D_\xi = \partial/\partial\xi$, $D_\eta = \partial/\partial\eta$

$$(D_\xi^2 + D_\eta^2)^2 A_0 Q^T = 0 \qquad (18)$$

$$(D_\xi^2 + D_\eta^2)^2 A_2 Q^T + \beta A_0 Q^T = 0 \qquad (19)$$

Let $-\frac{1}{\beta}H(\xi,\eta)$ be a solution of eq. (18), and let $P(\xi,\eta)$ be a particular integral of (19). Thus we have,

$$(D_\xi^2 + D_\eta^2)^2 P(\xi,\eta) = H(\xi,\eta) \qquad (20)$$

Let

$$(D_\xi^2 + D_\eta^2) P(\xi,\eta) = R(\xi,\eta) \qquad (21)$$

We can formally invert the above equations by defining the antiderivatives as $D_\xi^{-n}$ and $D_\eta^{-n}$. Combining eqs. (20) and (21) gives us

$$(D_\xi^2 + D_\eta^2) R(\xi,\eta) = H(\xi,\eta) \qquad (22)$$

$$R(\xi,\eta) = D_\xi^{-2}(1 + D_\xi^{-2} D_\eta^2)^{-1} H(\xi,\eta) \qquad (23)$$

$$= D_\xi^{-2}(1 - D_\xi^{-2} D_\eta^2 + D_\xi^{-4} D_\eta^4 - \ldots) H(\xi,\eta) \qquad (24)$$

To satisfy the twelve boundary conditions we can choose a simple bivariate cubic function, viz.,

$$A_0 Q^T = c_1 + c_2\xi + c_3\eta + c_4\xi^2 + c_5\xi^2\eta + \ldots + c_{12}\eta^3 \qquad (25)$$

Then $D_\eta^n H(\xi,\eta) = 0$ for $n > 3$, and only two terms remain in eq. (24), i.e.

$$R(\xi,\eta) = D_\xi^{-2} H(\xi,\eta) - D_\xi^{-4} D_\eta^2 H(\xi,\eta) \qquad (26)$$

and, similarly,

$$\begin{aligned} P(\xi,\eta) &= D_\xi^{-2} R(\xi,\eta) - D_\xi^{-4} D_\eta^2 R(\xi,\eta) \\ &= D_\xi^{-2} \left[ D_\xi^{-2} H(\xi,\eta) - D_\xi^{-4} D_\eta^2 H(\xi,\eta) \right] \\ &\quad - D_\xi^{-4} D_\eta^2 \left[ D_\xi^{-2} H(\xi,\eta) - D_\xi^{-4} D_\eta^2 H(\xi,\eta) \right] \\ &= D_\xi^{-4} H(\xi,\eta) - 2 D_\xi^{-6} D_\eta^2 H(\xi,\eta) \qquad (27) \end{aligned}$$

The last simplification comes from $D_\eta^n H(\xi,\eta) = 0$, $n > 3$ and $(D_\xi D_\eta - D_\eta D_\xi) * H(\xi,\eta) = 0$.

So the above represents a somewhat ad hoc procedure to find a particular integral, but obviously the answer is not unique, because we could have reversed the role of $D_\xi$ and $D_\eta$ at eq. (23) and/or at eq. (27). This freedom is however constrained by the physics of the problem which requires certain symmetry in the matrices K's and M's.

## SAMPLE OUTPUT

On the next two pages, we present some sample output from MACSYMA to indicate the complexity involved. We print the vectors $A_0$ and $A_2$ from eqs. (6) and (8), and $A_0$ from eq. (25). The matrices, however, are a bit too unwieldy to display for the present purpose. The two different $A_0$'s demonstrate that the fourth order problem is two orders of magnitude more complex than the second order problem (the vectors being one order and the matrices being two orders).

The two vectors on this page, given by eqs. (D5) and (D6) correspond to $A_0$ and $A_2$ from eqs. (6) and (8).

```
(C5) MATRIXMAP(FN,A0T);
                                  [ ETA XI - XI - ETA + 1 ]
                                  [                       ]
                                  [      XI - ETA XI      ]
(D5)                              [                       ]
                                  [        ETA XI         ]
                                  [                       ]
                                  [      ETA - ETA XI     ]

(C6) MATRIXMAP(FN,A2);
                     3           3     3      3       2    3      2           2     2   2
                    A  B ETA XI  + A B  ETA  XI    A  XI     A  ETA XI      A  XI
(D6) MATRIX([- ---------------------------------- + ------ + ----------- - ------
                         12 A B                        12          4             4

    2     2         2              2              2         2    3       2     2      2
   B  ETA  XI     B  ETA XI     A  ETA XI     A  XI     B  ETA      B  ETA      B  ETA
 + ----------- - ---------- - ----------- + ----- + -------- - -------- + ------,
        4            6              6            6        12          4           6

 3            3        3     3         2    3      2     2            2              2
A  B ETA XI  + A B  ETA  XI     A  XI      B  ETA  XI     B  ETA XI     A  ETA XI
---------------------------- - ------ - ----------- + ---------- - ----------
           12 A B                 12          4              6             12

    2              3                3    3          2            2
   A  XI      A  B ETA XI  + A B  ETA  XI     B  ETA XI     A  ETA XI
 + -----, - ------------------------------ + ---------- + ----------,
    12               12 A B                      12             12

 3            3        3     3         2        2     2              2              2     3
A  B ETA XI  + A B  ETA  XI     A  ETA XI      B  ETA XI     A  ETA XI      B  ETA
---------------------------- - ----------- - ---------- + ----------- - -------
           12 A B                   4             12              6             12

    2
   B  ETA
 + ------])
    12
```

The vector given below corresponds to $A_0$ from eq. (25).

```
(C4) MATRIXMAP(FN,AZ);
                            3        3               2          2             3             2
(D4) MATRIX([- 2 ETA XI  + 2 XI  + 3 ETA XI  - 3 XI  - 2 ETA  XI + 3 ETA  XI

                 3          2               3                2
 - ETA XI + 2 ETA  - 3 ETA  + 1], [- B ETA  XI + 2 B ETA  XI - B ETA XI

        3            2                          3        3               2          2
 + B ETA  - 2 B ETA  + B ETA], [A ETA XI  - A XI  - 2 A ETA XI  + 2 A XI

                                         3        3               2          2          3
 + A ETA XI - A XI], [2 ETA XI  - 2 XI  - 3 ETA XI  + 3 XI  + 2 ETA  XI

        2                        3                2
 - 3 ETA  XI + ETA XI], [B ETA  XI - 2 B ETA  XI + B ETA XI],

           3        3             2         2
[A ETA XI  - A XI  - A ETA XI  + A XI ],

             3              2            3              2
[- 2 ETA XI  + 3 ETA XI  - 2 ETA  XI + 3 ETA  XI - ETA XI],

       3               2                 2            3
[B ETA  XI - B ETA  XI], [A ETA XI  - A ETA XI ],

           3              2            3              2                       3         2
[2 ETA XI  - 3 ETA XI  + 2 ETA  XI - 3 ETA  XI + ETA XI - 2 ETA  + 3 ETA ],

         3              2              3          2
[- B ETA  XI + B ETA  XI + B ETA  - B ETA ],

             3                2
[- A ETA XI  + 2 A ETA XI  - A ETA XI])
```

# REFERENCES

1. Oliver, R. E., and Wilson, A. H.: Furlable Spacecraft Antenna Development: an interim report. Tech. Memo 33-537, Jet Propulsion Laboratory, Pasadena, California, 1972.

2. Gupta, K. K.: On a Finite Dynamic Element Method for Free Vibration Analysis of Structures. Computer Methods in Applied Mechanics and Engineering, vol. 9, pp. 105-120, 1976.

BLANK PAGE

BLANK PAGE

# SYMBOLIC MANIPULATION TECHNIQUES FOR VIBRATION ANALYSIS OF LAMINATED ELLIPTIC PLATES*

C. M. Andersen
The College of William and Mary in Virginia

Ahmed K. Noor
The George Washington University
Joint Institute for Advancement of Flight Sciences
at NASA Langley Research Center

## SUMMARY

A computational scheme is presented for the free vibration analysis of laminated composite elliptic plates. The scheme is based on Hamilton's principle, the Rayleigh-Ritz technique and symmetry considerations and is implemented with the aid of the MACSYMA symbolic manipulation system. The MACSYMA system, through differentiation, integration and simplification of analytic expressions, produces highly-efficient FORTRAN code for the evaluation of the stiffness and mass coefficients. Multiple use is made of this code to obtain not only the frequencies and mode shapes of the plate, but also the derivatives of the frequencies with respect to various material and geometric parameters.

## INTRODUCTION

Many of the boundary-value problems which arise in engineering and physics cannot be solved in a closed or analytic form. Therefore, numerical methods are necessary for their solution. Nevertheless, we can expect that some of the steps in the solution process will be symbolic or analytic in nature. For example, early steps in the solution process may involve (a) casting the governing differential or functional equations in a more convenient form for solution through replacement of the fundamental unknowns by new variables which are dimensionless or have other desirable properties, and (b) the introduction of approximation functions or perturbation expansions and a regrouping of the various terms. Thus, the solution process can be thought of as consisting of a symbolic (or analytic) phase followed by a numerical phase. With the aid of computerized algebraic manipulation, we may sometimes carry the symbolic phase of the calculation further than is conventionally done and thereby reduce the cost and/or improve the accuracy of the calculations.

---

*Work supported by NASA Langley Research Center.

A case in point is the free vibration analysis of laminated composite elliptic plates (refs. 1 and 2). A plate is a flat body whose thickness is small compared to its other dimensions. Plates and other structures formed from *composite materials* such as graphite or boron fibers imbedded in a matrix of epoxy or polyimide resins have considerable interest to the aircraft industry because of their high strength and rigidity, easy machinability and light weight. These composites are characterized by extremely high tensile strength in the direction of the fibers but relatively low strength in directions normal to the fibers. As a consequence, the composites are typically used in laminated structures where the orientation of the fibers changes from lamina to lamina. The highly anisotropic behavior of composite materials considerably complicates the analysis of the structures in which they are used. An investigation of the dependence of the frequencies of vibration (and the associated mode shapes) on the various geometric and lamination parameters is needed for the efficient design of plates made from composite materials. This requires not only the efficient evaluation of the frequencies and mode shapes for a given set of parameters, but also the efficient computation of the derivatives of the frequencies with respect to the various design variables. Such derivatives provide information about the sensitivity of the frequencies to changes in the design variables.

The objectives of the present paper are to develop a computational scheme for the free vibration analysis of laminated composite elliptic plates with clamped edges and to identify the major advantages gained from the use of symbolic manipulation in the solution process. The main elements of the scheme include (1) the use of the Rayleigh-Ritz method in conjunction with Hamilton's principle, (2) simplification of the computation through considerations of various types of symmetries, (3) the use of the MACSYMA symbolic manipulation system to generate efficient FORTRAN code, and (4) multiple use of that code in the determination of both frequencies and frequency derivatives. Because of the elliptical shape of the plates, MACSYMA is able to provide short exact analytic forms for a large number of expressions which would otherwise have to be approximated through the use of numerical quadrature.

## MATHEMATICAL FORMULATION

Figure 1 shows an elliptic plate and its Cartesian coordinate system. The z-axis is normal to the flat surfaces of the plate, and the x- and y-axes lie in the middle plane along the principal axes of the ellipse. The problem domain is thus specified by

$$(x/a)^2 + (y/b)^2 \leq 1 \qquad -h/2 \leq z \leq h/2 \tag{1}$$

In this study we treat the plate vibration problem as a three-dimensional elasticity problem. A free vibration mode of the plate is described by a frequency $\omega$ (actually an angular velocity) and by the displacement amplitudes $u_i(x,y,z)$ $(i = 1,2,3)$. A point in the vibrating plate with equilibrium position $(x,y,z)$ will have the position

$\left(x+u_1(x,y,z)\sin\omega t,\ y+u_2(x,y,z)\sin\omega t,\ z+u_3(x,y,z)\sin\omega t\right)$ at time $t$.

The components $\varepsilon_{ij}(x,y,z)$ of the strain tensor are defined in terms of the displacement components $u_j$ by

$$\varepsilon_{ij}(x,y,z) = \frac{1}{2}\left(\partial_i u_j + \partial_j u_i\right) \quad (i,j = 1,2,3) \tag{2}$$

where $\partial_1 = \partial/\partial x$, $\partial_2 = \partial/\partial y$ and $\partial_3 = \partial/\partial z$. We group the six strain components into a vector $\bar{\varepsilon}_I(x,y,z)$ $(I = 1\to 6)$ by letting

$$\begin{array}{lll} \bar{\varepsilon}_1 = \varepsilon_{11} & \bar{\varepsilon}_2 = \varepsilon_{22} & \bar{\varepsilon}_3 = \varepsilon_{33} \\ \bar{\varepsilon}_4 = 2\,\varepsilon_{23} & \bar{\varepsilon}_5 = 2\,\varepsilon_{31} & \bar{\varepsilon}_6 = 2\,\varepsilon_{12} \end{array} \tag{3}$$

We analogously define a stress vector $\bar{\sigma}_I(x,y,z)$ $(I = 1\to 6)$ in terms of the six independent components of the stress tensor and assume the stress-strain relationship is linear and given by the constitutive relation

$$\bar{\sigma}_I(x,y,z) = \sum_{J=1}^{6} c_{IJ}(z)\,\bar{\varepsilon}_J(x,y,z) \quad (I = 1\to 6) \tag{4}$$

We assume that $c_{IJ}(z)$ is constant within each layer but can vary from layer to layer. Further, we assume that the fibers are all parallel to the x-y plane. As a consequence, the $c_{IJ}(z)$ form a symmetric matrix of the form

$$[c(z)] = \begin{bmatrix} c_{11}(z) & c_{12}(z) & c_{13}(z) & 0 & 0 & c_{16}(z) \\ & c_{22}(z) & c_{23}(z) & 0 & 0 & c_{26}(z) \\ & & c_{33}(z) & 0 & 0 & c_{36}(z) \\ & & & c_{44}(z) & c_{45}(z) & 0 \\ & & & & c_{55}(z) & 0 \\ \text{Symmetric} & & & & & c_{66}(z) \end{bmatrix}$$

The strain energy $U$ and the kinetic energy $T$ are given in terms of the strains and displacements by

$$U = \frac{1}{2} \int \sum_{I,J=1}^{6} \bar{\varepsilon}_I(x,y,z)[c_{I,J}(z)\, \bar{\varepsilon}_J(x,y,z)]\, dx\, dy\, dz$$

$$T = \frac{\omega^2}{2} \int \rho(z) \sum_{i=1}^{3} [u_i(x,y,z)]^2\, dx\, dy\, dz \qquad (6)$$

where $\rho(z)$ is the density of the plate material. Since we assume that $\rho(z)$, like $[c(z)]$, is constant within each layer but can vary from layer to layer, the integrations in the z-direction are to be performed in a piecewise manner.

The quantity $\Pi(u_i) = T - U$ is to be regarded as a functional of the displacement functions $u_i(x,y,z)$. Hamilton's variational principle states that $u_i(x,y,z)$ must be such that the quantity $\Pi(u_i)$ is stationary with respect to variations in the displacement functions, i.e.

$$\delta\Pi = 0 \qquad (7)$$

where $\delta\Pi$ is the symbol for the first variation of $\Pi$. The variational principle thus gives rise to a set of elliptic partial differential equations in the $u_i(x,y,z)$. However, rather than explicitly developing these differential equations we shall adopt a slightly different approach. We approximate the $u_i(x,y,z)$ in $\Pi$ by linear combinations over a set of approximation functions. The coefficients $\psi_j$ $(j = 1 \rightarrow N)$ which appear in these linear combinations are determined from the requirement that

$$\frac{\partial\Pi(\psi_i)}{\partial\psi_i} = 0 \qquad (i = 1 \rightarrow N) \qquad (8)$$

This results in a linear generalized eigenvalue problem of the form

$$\sum_{j=1}^{N} K_{ij}\, \psi_j = \omega^2 \sum_{j=1}^{N} M_{ij}\, \psi_j \qquad (9)$$

**where**

$$K_{ij} = \frac{1}{\pi} \frac{\partial^2 U}{\partial\psi_i\, \partial\psi_j}$$

$$M_{ij} = \frac{1}{\omega^2} \frac{\partial^2 T}{\partial\psi_i\, \partial\psi_j} \qquad (10)$$

## SYMBOLIC PHASE OF COMPUTATION

The first step in the symbolic phase of computation is to approximate the displacements $u_i(x,y,z)$ in the functional $\Pi$. The boundary conditions

$$u_i(x,y,z) = 0 \qquad (i = 1,2,3) \tag{11}$$

along the clamped edge of the plate are automatically satisfied by the use of approximations of the form

$$u_i(x,y,z) = \sum_k z^k \sum_{m,n} \phi_i^{m,n,k} \, [1-(x/a)^2 - (y/b)^2] \, x^m y^n \tag{12}$$

where the upper limit of $k$ in the summation is one higher for $i=1$ or $2$ than for $i=3$. The number of terms needed in the expansion (12) depends on the thickness of the plate as well as on the accuracy desired for the solutions. The $\psi_j$ of eqs. (8) through (10) are the coefficients $\phi_i^{m,n,k}$ taken in some arbitrary order. The symbolic phase of the computation can proceed as follows:

(1) Select a (new) pair of indices, $i$ and $j$, for which expressions for $K_{ij}$ and $M_{ij}$ are desired (see the section on Symmetry Considerations).

(2) Set all $\psi_k$ to zero except $\psi_i$ and $\psi_j$, which remain as undefined (atomic) variables.

(3) Form the terms of $u_k(x,y,z)$ $(k = 1,2,3)$ which depend on $\psi_i$ and $\psi_j$ using eq. (12).

(4) Compute the terms of $\varepsilon_I(x,y,z)$ $(I = 1 \to 6)$ which depend on $\psi_i$ and $\psi_j$ using eq. (2).

(5) Evaluate the terms of the integrands of $U$ and $T$ which depend on $\psi_i$, $\psi_j$ using eq. (6).

(6) Evaluate the integrands of $K_{ij}$ and $M_{ij}$ by differentiation with respect to both $\psi_i$ and $\psi_j$ using eq. (10).

(7) Evaluate $K_{ij}$ and $M_{ij}$ by performing the integrations over $x,y$ and $z$ via pattern matching.

(8) Simplify the nonzero $K_{ij}$ and $M_{ij}$ and develop FORTRAN expressions for them.

(9) Go to step (1) unless finished.

In step (7), the integration with respect to the $z$ coordinate is accomplished symbolically (analytically) simply by introducing new variables $C_{IJ}^{(\ell)}$ and

$D^{(\ell)}$ defined by

$$C_{IJ}^{(\ell)} = \int_{-h/2}^{h/2} z^{\ell}\, C_{IJ}(z)\, dz \qquad (I,J = 1 \rightarrow 6;\ \ \ell = 0,1,2 \ldots)$$
$$D^{(\ell)} = \int_{-h/2}^{h/2} z^{\ell}\, \rho(z)\, dz \qquad (\ell = 0,1,2 \ldots) \tag{13}$$

and the integration with respect to $x$ and $y$ is accomplished by the replacements

$$x \rightarrow a\ r\ \cos(\theta)$$
$$y \rightarrow b\ r\ \cos(\theta) \tag{14}$$

followed by exact closed-form integration in $r$ and $\theta$. The expressions produced for $K_{ij}$ and $M_{ij}$ in step (8) are very simple since an $M_{ij}$ expression contains at most a single term, and a $K_{ij}$ expression contains at most three terms. The $M_{ij}$ are linear in the $D^{(\ell)}$ and the most general form for the $K_{ij}$ is

$$K_{ij} = \begin{cases} \lambda_1 \\ \lambda_2 A \\ \lambda_3 B \\ (\lambda_4\ A2 + \lambda_5\ B2 + \lambda_6\ A2B2)/(\lambda_7\ AB) \end{cases} \tag{15}$$

where $\lambda_1$, $\lambda_2$ and $\lambda_3$ are linear combinations of the $C_{IJ}^{(\ell)}$ with rational number coefficients; $\lambda_4$, $\lambda_5$ and $\lambda_6$ are integer multiples of the $C_{IJ}^{(\ell)}$; $\lambda_7$ is an integer; and the FORTRAN variables A, B, AB, A2, B2, A2B2 are defined by

$$A = a \qquad B = b \qquad AB = a\ b$$
$$A2 = a^2 \qquad B2 = b^2 \qquad A2B2 = a^2 b^2 \tag{16}$$

The symbolic phase ends when the FORTRAN code has been transferred to a local computer for the numerical phase of computation.

## NUMERICAL PHASE OF COMPUTATION

The first goal of the numerical phase of computation is to solve the linear generalized eigenvalue problem (eq. (9)) for the lowest few frequencies $\omega_k (k=1,2, \ldots)$. To accomplish this the numerical program evaluates the $C_{IJ}^{(\ell)}$, the $\rho^{(\ell)}$, the FORTRAN variables of eq. (16), and finally the $K_{ij}$ and $M_{ij}$. Then the eigenvalues $(\omega_k)^2$ and their associated eigenvectors $\psi_j$ may be determined by the method of subspace iteration (ref. 3).

The second goal of the numerical phase of computation is to determine the derivatives of the $\omega_k$ with respect to changes in geometry, fiber orientations or material properties. The derivative of the frequency $\omega_k$ with respect to the plate area $\pi ab$ (keeping the aspect ratio $a/b$, thickness $h$ and material properties fixed) is given by

$$\left.\frac{\partial\omega_k}{\partial(\pi ab)}\right|_{h,\ a/b} = \frac{-\displaystyle\sum_{i,j=1}^{N} \psi_i^{(k)} \left.\frac{\partial(\pi ab\ K_{ij})}{\partial(\pi ab)}\right|_{h,\ a/b} \psi_j^{(k)}}{2\omega_k(\pi ab)\left(\displaystyle\sum_{i,j=1}^{N} \psi_i^{(k)} M_{ij}\ \psi_j^{(k)}\right)} \tag{17}$$

This equation takes into account the fact that each $M_{ij}$ is proportional to the area but is independent of the aspect ratio. The derivative on the RHS of eq. (17) is evaluated by using the FORTRAN code for the $K_{ij}$ but with FORTRAN variables of eq. (16) defined as follows:

$$\begin{array}{lll} A = a/2 & B = b/2 & AB = 1 \\ A2 = a/b & B2 = b/a & A2B2 = 0 \end{array} \tag{18}$$

The computational effort involved in the evaluation of $\frac{\partial\omega}{\partial(\pi ab)}$ using eq. (17) is considerably less than that required for solving the eigenvalue problem. Note that it would be difficult to evaluate the derivative matrix in eq. (17) by conventional numerical techniques. The derivative of $\omega_k$ with respect to the thickness $h$ (keeping $a$ and $b$ fixed) is given by

$$\left.\frac{\partial\omega_k}{\partial h}\right|_{a,b} = -\frac{1}{h}\left[\omega_k + 2(\pi ab)\left.\frac{\partial(\omega_k)}{\partial(\pi ab)}\right|_{h,\ a/b}\right] \tag{19}$$

This equation is based on the fact that the replacement of $a$ by $\lambda a$, $b$ by $\lambda b$, and $h$ by $\lambda h$ (keeping the relative thicknesses of the laminae constant) results in $\omega_k$ being replaced by $\lambda^{-2}\omega_k$. The derivative of the frequency $\omega_k$ with respect to a change in the aspect ratio $a/b$ (keeping the area $\pi ab$ and thickness $h$ fixed) is given by

$$\left.\frac{\partial\omega_k}{\partial(a/b)}\right|_{h,\ ab} = \frac{\displaystyle\sum_{i,j=1}^{N} \psi_i^{(k)} \left.\frac{\partial(K_{ij})}{\partial(a/b)}\right|_{h,\pi ab} \psi_j^{(k)}}{2\omega_k\left(\displaystyle\sum_{i,j}^{N} \psi_i^{(k)} M_{ij}\ \psi_j^{(k)}\right)} \tag{20}$$

where the summation in the denominator is the same as in eq. (17). We now need to make the $\lambda_1$ and $\lambda_6$ terms of eq. (15) vanish since they do not depend on $a/b$. We accomplish this by setting

$$\begin{aligned} &A = N\ b/2 \qquad && B = -\ N\ b^2/(2a) \qquad && AB = a\ b \\ &A2 = N\ a\ b \qquad && B2 = -\ N\ b^3/a \qquad && A2B2 = 0 \end{aligned} \tag{21}$$

where $N$ is a very large number (e.g., $N = 10^{15}$), and compensate for the introduction of $N$ by dividing by $N$ after the summation indicated in eq. (20) has been carried out. When the derivatives of $\omega_k$ with respect to area, aspect ratio and thickness are known, one can easily determine the derivatives of $\omega_k$ with respect to $a$, $b$ and/or any other functions of $ab$, $a/b$ and $h$.

Derivatives of $\omega_k$ with respect to the fiber orientation angles or material properties may be computed similarly, but for these cases the FORTRAN variables of eq. (21) regain their original definitions (eq. (16)) and the $\lambda_i$ $(i = 1 \rightarrow 6)$ are replaced by their appropriate derivatives. This kind of multiple use of a large block of FORTRAN code is very useful for reducing the length of the FORTRAN program as well as the amount of symbolic computation. Both are further reduced by the symmetry considerations discussed in the next section.

## SYMMETRY CONSIDERATIONS

There are three types of symmetries which help simplify our calculations. These are associated with a) symmetry of the [K] and [M] matrices, b) rotation-reflection symmetry of the undeformed plate, and c) symmetry of the stiffness and mass coefficients with respect to interchanging the roles of $a$, $b$ and the subscripts 1,2.

### Symmetry of the [K] and [M] Matrices

The first type of symmetry is the symmetry of the [K] and [M] matrices under transposition, that is

$$\begin{aligned} K_{ij} &= K_{ji} \\ M_{ij} &= M_{ji} \end{aligned} \tag{22}$$

(see eq. (10)). The presence of this symmetry means that we need symbolic expressions only for those $K_{ij}$ and $M_{ij}$ with $i \leq j$.

## Rotation-Reflection Symmetry of the Undeformed Plate

The second type of symmetry is the symmetry of the (undeformed) plate itself. Various rotations or reflections may leave the boundaries and material properties of the plate invariant (ref. 4). For instance, by our assumption that the fiber directions are parallel to the plate, rotations of the plate by $180^{\circ}$ about the $z$-axis leave $[\bar{c}(z)]$ invariant.

A consequence of this symmetry (the symmetry group is called $C_2$ in Schoenflies notation (ref. 5)) is that there are two families of solutions - those with $\sigma = 1$ and those with $\sigma = -1$ in the relations

$$\begin{aligned} u_1(x,y,z) &= -\sigma\, u_1(-x,-y,z) \\ u_2(x,y,z) &= -\sigma\, u_2(-x,-y,z) \\ u_3(x,y,z) &= \sigma\, u_3(-x,-y,z) \end{aligned} \qquad (23)$$

Equation (23) defines the minimum symmetry exhibited by the laminated plates considered in the present study.

The largest symmetry group which can leave the boundaries invariant is the group $D_{2h}$. A plate which has this symmetry is invariant under rotations by $180^{\circ}$ not only around the z-axis but around the $x$- and $y$-axes as well. Further, it is invariant under reflections in the $x$-$y$, $y$-$z$ and $z$-$x$ planes and under inversion (the operation which sends the generic point $(x,y,z)$ to the point $(-x,-y,-z)$). Plates with $D_{2h}$ symmetry have eight families of solutions each corresponding to one of the possible combinations of $\sigma_1 = \pm 1$, $\sigma_2 = \pm 1$, $\sigma_3 = \pm 1$ in the relations

$$\begin{aligned} u_1(x,y,z) &= -\sigma_1 u_1(-x,y,z) = \sigma_2 u_1(x,-y,z) = -\sigma_3 u_1(x,y,-z) \\ u_2(x,y,z) &= \sigma_1 u_2(-x,y,z) = -\sigma_2 u_2(x,-y,z) = -\sigma_3 u_2(x,y,-z) \\ u_3(x,y,z) &= \sigma_1 u_3(-x,y,z) = \sigma_2 u_3(x,-y,z) = \sigma_3 u_3(x,y,-z) \end{aligned} \qquad (24)$$

For the four families with $\sigma_3 = -1$ the middle surface of the plate (the surface with $z = 0$) is deformed with planar motions only. In order for a laminated composite plate to have the full $D_{2h}$ symmetry, the fiber angle with respect to the $x$-axis, $\theta(z)$, must take only the values $0^{\circ}$ and $90^{\circ}$ and $\theta(z)$ must equal $\theta(-z)$.

The group $D_{2h}$ has three subgroups of order four which contain $C_2$ as a subgroup. In Schoenflies notation they are called $C_{2h}$, $C_{2v}$ and $D_2$. Each of these subgroups correspond to a possible plate symmetry higher than the minimal $C_2$ symmetry yet lower than the full $D_{2h}$ symmetry. Plates with any of these symmetries have four families of solutions. Plates with symmetry $C_{2h}$ have $\theta(z)$ equal to $\theta(-z)$ and have solutions characterized by $(\sigma, \sigma_3) = (1,1)$, $(1,-1)$, $(-1,1)$ or $(-1,-1)$ in

$$
\begin{aligned}
u_1(x,y,z) &= -\sigma\ u_1(-x,-y,z) = -\sigma_3 u_1(x,y,-z) \\
u_2(x,y,z) &= -\sigma\ u_2(-x,-y,z) = -\sigma_3 u_2(x,y,-z) \\
u_3(x,y,z) &= \sigma\ u_3(-x,-y,z) = \sigma_3 u_3(x,y,-z)
\end{aligned}
\tag{25}
$$

Plates with symmetry $C_{2v}$ have fiber angles of $0^{\circ}$ and $90^{\circ}$ only and have solutions characterized by $(\sigma_1, \sigma_2) = (1,1), (1,-1), (-1,1)$ or $(-1,-1)$ in

$$
\begin{aligned}
u_1(x,y,z) &= -\sigma_1 u_1(-x,y,z) = \sigma_2 u_2(x,-y,z) \\
u_2(x,y,z) &= \sigma_1 u_2(-x,y,z) = -\sigma_2 u_2(x,-y,z) \\
u_3(x,y,z) &= \sigma_1 u_3(-x,y,z) = \sigma_3 u_3(x,-y,z)
\end{aligned}
\tag{26}
$$

Plates with symmetry $D_2$ are invariant under rotations by $180^{\circ}$ about the x-, y- and z-axes and thus have

$$
\theta(z) = -\theta(-z); \quad -90^{\circ} \leq \theta \leq 90^{\circ}
\tag{27}
$$

For these plates we let

$$
u_i(x,y,z) = u_i^e(x,y,z) + u_i^o(x,y,z)
\tag{28}
$$

where

$$
\begin{aligned}
u_i^e(x,y,z) &= u_i^e(x,y,-z) \\
u_i^o(x,y,z) &= -u_i^o(x,y,-z)
\end{aligned}
\tag{29}
$$

Then the solutions are characterized by $(\sigma_1, \sigma_2) = (1,1), (1,-1), (-1,1)$ or $(-1,-1)$ in

$$
\begin{aligned}
u_1^e(x,y,z) &= \sigma_1 u_1^e(-x,y,z) = -\sigma_2 u_1^e(x,-y,z) \\
u_2^e(x,y,z) &= -\sigma_1 u_2^e(-x,y,z) = \sigma_2 u_2^e(x,-y,z) \\
u_3^e(x,y,z) &= \sigma_1 u_3^e(-x,y,z) = \sigma_2 u_3^e(x,-y,z)
\end{aligned}
\tag{30}
$$

with $(\sigma_1, \sigma_2)$ replaced by $(-\sigma_1, -\sigma_2)$ in the corresponding relations for $u_i^o$. If any two of the eqs. (25), (26) and (30) hold simultaneously then eq. (24) must hold. On the other hand, eq. (23) is a consequence of eqs. (25), (26) or (30) separately.

Solutions lacking the appropriate symmetry are possible only in the (unlikely) event that the eigenvalues for members of two different families of solutions coincide, in which case the solutions are linear combinations of symmetric solutions. The presence of families of solutions with different symmetries means that with the choice of a proper ordering of the $\psi_i$ the [K] and [M] matrices have a block diagonal form with one block for each family of solutions $\tau$. That is, $\Pi$ may be written as

$$\Pi = \sum_{\tau} \Pi_{\tau} \tag{31}$$

where $\Pi_\tau$ contains the $\psi_i$ associated with the symmetry $\tau$. This results in replacing a large problem by two, four or eight (depending on the symmetry group) smaller subproblems. For each of the subproblems, the expansion in eq. (12) is adjusted to match the desired symmetries.

## Symmetry of Stiffness and Mass Coefficients With Respect to Interchanging the Roles of a,b and the Subscripts 1,2

The third type of symmetry is related to the observation that when given a physical plate we may analyze it in two different ways - with the semi-major axis of the plate along the x-axis or along the y-axis. The two ways are equivalent but result in interchanging the numerical values for a and b and for some of the material properties $\bar{c}_{IJ}(z)$. Let $K_{ij}$ and $K^*_{ij}$ be components of the stiffness matrices (before the partitioning of eq. (31)) for the same physical problem as formulated in the two different ways. While it is not true in general that $K_{ij}$ equals $K^*_{ij}$, it is true that for each pair of indices i,j there corresponds a pair i', j' such that $K_{ij} = K^*_{i'j'}$; thus

$$K_{i'j'}(a,b,C^{(\ell)}_{11}, C^{(\ell)}_{13}, C^{(\ell)}_{16}, C^{(\ell)}_{22}, C^{(\ell)}_{23}, C^{(\ell)}_{26}, \ldots)$$

$$= K_{ij}(b,a,C^{(\ell)}_{22}, C^{(\ell)}_{23}, C^{(\ell)}_{26}, C^{(\ell)}_{11}, C^{(\ell)}_{13}, C^{(\ell)}_{16}, \ldots) \tag{32}$$

Thus, while $K_{ij}$ and $K_{i'j'}$ do not necessarily have the same numerical value, they do have essentially the same algebraic form, and the FORTRAN code used to evaluate $K_{ij}$ can serve to evaluate $K_{i'j'}$ as well. The relation turns out to be even stronger for the [M] matrix since

$$M_{ij} = M_{i'j'} \tag{33}$$

The first, second and third types of symmetries interact with each other in the following way. Either all the index pairs i,j in the block of the [K] matrix associated with symmetry $\tau$ correspond to index pairs i',j' in the block having a different symmetry $\tau'$ or they all correspond to i',j' in the same block. For the former case the FORTRAN code generated to find the solutions with symmetry $\tau$ can be used to find the solutions with symmetry $\tau'$ as well. For the latter case the relations (22), (32) and (33) together serve to reduce the FORTRAN code needed for symmetry $\tau$ to little more than half that needed when considering eq. (22) alone. For this case the code is executed once and the incomplete [K] and [M] saved. Then the code is executed a second time with variables interchanged as in eq. (32) and the two sets of matrices are merged. The interactions of the three types of symmetry are summarized in Table 1 for the five symmetry groups of interest. The symmetry considerations discussed in this section apply equally well for the determination of the derivative matrices in eqs. (17), (19) and (20).

TABLE 1. - INTERACTIONS AMONG THE FIRST, SECOND AND THIRD TYPES OF SYMMETRIES

| Symmetry Group of Plate | Symmetry Parameters, $\tau$ | Symmetries for which [K] and [M] are simplified by eqs. (22), (32) and (33) | Symmetries interrelated by eqs. (22), (32) and (33) |
|---|---|---|---|
| $C_2$ | $(\sigma)$ | (1), (-1) | |
| $C_{2h}$ | $(\sigma, \sigma_3)$ | (1,1), (1,-1), (-1,1), (-1,-1) | |
| $C_{2v}$ | $(\sigma_1, \sigma_2)$ | (1,1), (-1,-1) | (1,-1) ↔ (-1,1) |
| $D_2$ | $(\sigma_1, \sigma_2)$ | (1,1), (-1,-1) | (1,-1) ↔ (-1,1) |
| $D_{2h}$ | $(\sigma_1, \sigma_2, \sigma_3)$ | (1,1,1), (1,1,-1),<br>(-1,-1,1), (-1,-1,-1) | (1,-1,1) ↔ (-1,1,1);<br>(1,-1,-1) ↔ (-1,1,-1) |

## NUMERICAL RESULTS

Numerical results have been obtained for moderately thick laminated plates with symmetry $D_2$. For the case $\sigma_1 = \sigma_2 = 1$, we use the following version of eq. (12) which takes eq. (30) into account:

$$u_1(x,y,z) = [1-(\frac{x}{a})^2-(\frac{y}{b})^2][y\chi_1(x,y) + xz\chi_4(x,y) + yz^2\chi_7(x,y) + xz^3\chi_{10}(x,y)]$$

$$u_2(x,y,z) = [1-(\frac{x}{a})^2-(\frac{y}{b})^2][x\chi_2(x,y) + \chi_5(x,y) + xz^2\chi_8(x,y) + yz^3\chi_{11}(x,y)]$$

$$u_3(x,y,z) = [1-(\frac{x}{a})^2-(\frac{y}{b})^2][\chi_3(x,y) + xyz\chi_6(x,y) + z^2\chi_9(x,y)] \qquad (34)$$

where

$$\psi_i(x,y) = \psi_i + \psi_{11+i}\, x^2 + \psi_{22+i}\, y^2 + \psi_{33+i}\, x^2 y^2 + \psi_{44+i}\, x^4 + \psi_{55+i}\, x^6$$

$$+ \psi_{66+i}\, x^4 y^2 + \psi_{77+i}\, x^2 y^4 + \psi_{88+i}\, x^6 + \psi_{99+i}\, y^6 \quad (i = 1 \rightarrow 11) \qquad (35)$$

This approximation scheme results in matrices [K] and [M] having dimension 110 by 110 and requires the generation of 3541 FORTRAN statements. Similar approximation schemes are used for the other families of solutions. Typical results are shown in figure 2. These results are for eight-layered plates with $h = b/10$ and fiber orientations (with respect to the x-axis) which are alternately $\theta$ and $-\theta$, where $\theta = 45^0$. The material properties are chosen to be those typical of a high-modulus graphite-epoxy composite. Figure 2 shows the variation with the aspect ratio $a/b$ of the lowest frequencies and of the derivatives of these frequencies with respect to the fiber orientation angle $\theta$.

## CONCLUDING REMARKS

The major advantages of using symbolic manipulation in the free vibration analysis of laminated composite elliptic plates are

1) The accurate and reliable symbolic evaluation of large numbers of derivatives and integrals

2) The concise form of the resulting FORTRAN expressions for $K_{ij}$ and $M_{ij}$

3) The ease of implementing symmetry concepts

4) The simplicity of evaluating the first derivatives of the frequencies with respect to the design variables

The multiple usage of the large blocks of FORTRAN code generated by MACSYMA allows the calculation of frequency derivatives with no extra symbolic effort and very little extra numerical computation. Of course, the symbolic approach would be useless were it not for the fact that the output is in the form of FORTRAN statements which need never be keypunched. Manual operations on such a large quantity of data would surely introduce errors which would be very difficult to rectify.

The major disadvantages are

1) The large amount of FORTRAN code needed to obtain accurate numerical results

2) The relatively long symbolic computation times

3) The slow speed of transferring data from the symbolic processing computer to the number processing computer when the two computers are not on the same network

Several extensions of the present work come to mind, such as studying plates with other boundary conditions and other geometries. Shapes requiring numerical quadrature for the x-y integration may also be investigated. The various integrals required can be identified, isolated and assigned variable names through the use of symbol manipulation much as the z-integrals are treated in the present study. The techniques used herein are applicable to a wide variety of other boundary-value problems.

## REFERENCES

1. Andersen, C. M.; and Noor, Ahmed K.: Free Vibration of Laminated Composite Elliptic Plates. Advances in Engineering Science, Volume 2, NASA CP-2001, 1976, pp. 425-438.

2. Ashton, J. E.; and Whitney, J. M.: Theory of Laminated Plates. Technomic Publ. Co., 1970.

3. Bathe, K. J.; and Wilson, E. L.: Numerical Methods in Finite Element Analysis. Prentice-Hall, Inc., 1976.

4. Noor, A. K.: Symmetries in Laminated Composite Plates. Proceedings of the Eighth Southeastern Conference on Theoretical and Applied Mechanics, Virginia Polytech. Inst. & State Univ., Apr. 1976, pp. 225-246.

5. Hamermesh, Morton: Group Theory and Its Application to Physical Problems. Addison-Wesley Pub. Co., Inc., c.1962.

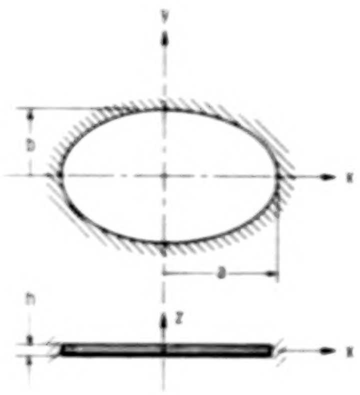


Fig. 1. Clamped laminated elliptic plate.

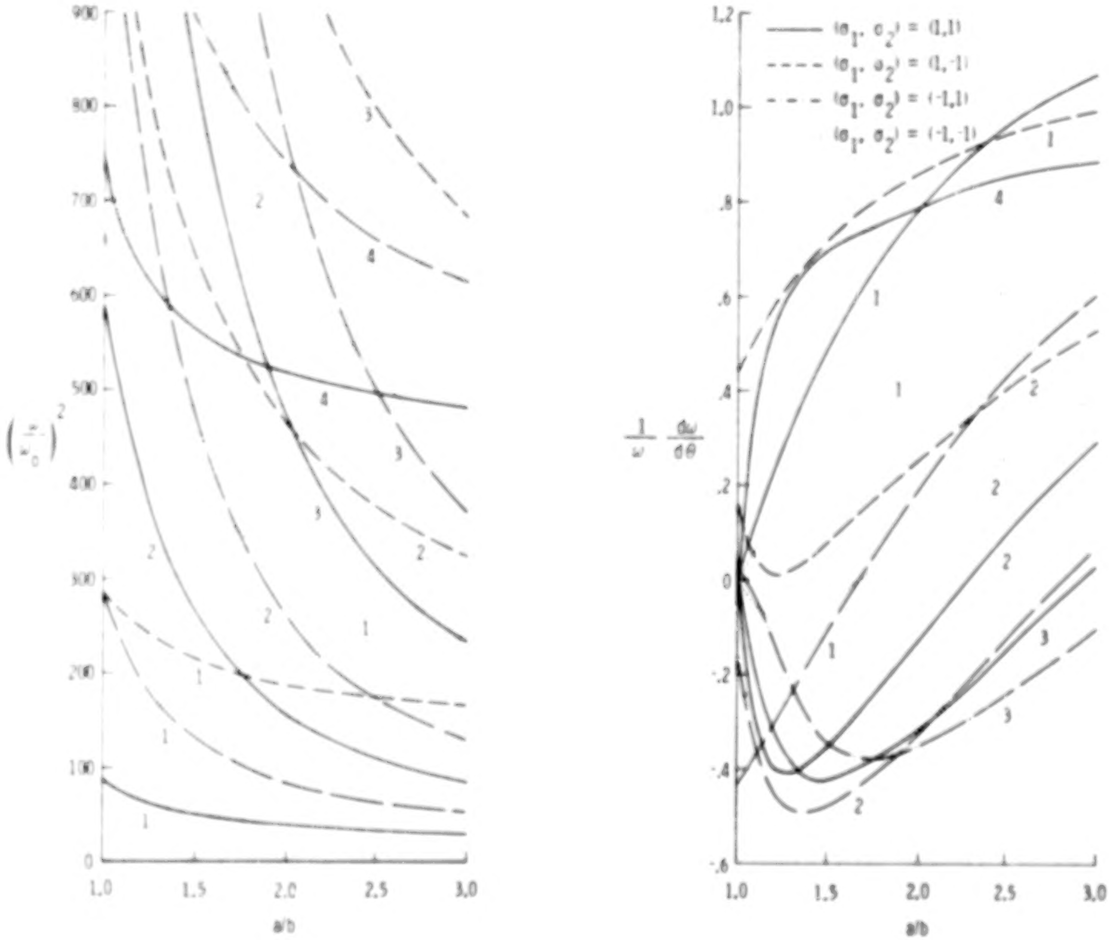


Fig. 2. Effect of a/b on $\omega^2$ and $\frac{1}{\omega}\frac{d\omega}{d\theta}$ for clamped eight-layered elliptic plates with $D_2$ symmetry and fiber orientations alternately +45° and -45°. $h/b = 1/10$; $E_L/E_T = 40$; $\nu_{LT} = 1/4$; $G_{LT}/E_T = 3/5$; $G_{TT}/E_T = 1/2$; $\omega_0^2 = (E_T h^2)/(\rho b^4)$.

BLANK PAGE

BLANK PAGE

# OBSERVATIONS ON APPROXIMATE INTEGRATIONS

Edward W. Ng
Jet Propulsion Laboratory

## Extended Abstract

In this presentation we explore a class of integration strategies that fall in between the two extremes of symbolic integration and numerical quadrature, which are, respectively, aimed at the computer generation of answers in the form of exact expressions and numerical values. We shall first discuss the theoretical advances in symbolic integration, as motivation to the following, then examine three major contexts of applications with attendant case studies, and finally explore four possible types of strategies for approximate integration. In particular we shall comment on the feasibility and adequacy (or inadequacy) of MACSYMA for implementing these strategies.

We begin with theoretical discussions. In this aspect we have discerned two major paradigms of strategies, which we label the "pattern-recognition paradigm" and the "problem-solving paradigm". These labels, though far from perfect, are chosen to indicate the emphasis only. In the former class we include, for example, Risch's algorithm, (Ref. 1) and Moses' new approach based on extension operators (Ref. 2). We believe these strategies to be particularly characterized by the search of algorithmic ability to recognize that certain expressions or operators belong to some specified class of such. The problem solving paradigm is obviously inherited from heuristic strategies of artificial intelligence. In this latter class we include, for example, Wang's definite integrations (Ref. 3) and our elliptic integrations (Ref. 4). All these theoretical strategies suffer from practical limitations of one kind or another. Notably among these are the multivariate factorization problem, the optimal selection of input vis-a-vis output class of expressions and intelligent choice of contours for definite integration. The optimal selection needs particular elaboration here. Take for example the integration of rational functions. It is easy to devise an efficient algorithm to decide if a given rational function can be integrated in terms of rational functions. But such algorithm would be of extremely limited interest because it would return a negative answer for most input expressions, such as something as simple as 1/(x+1). The addition of one 'new' function (logarithm) in the output class dramatically expands the problem-solving horizon. On the other hand, we obviously cannot carry this to the other extreme of choosing a large number of new functions, lest the result be next to worthless. All these discussions, however, force us to consider what we mean by 'usefulness' of an output expression, which in turn leads us to considering three major contexts of applications.

At this Laboratory we have been associated with an applied mathematics group which provides consultation and support to a diversity of engineers and scientists. Although our picture is still somewhat limited, it does give us an

indication of the major contexts in which integration tools are considered necessary or useful. The first is the usual exploratory context, where a scientist or engineer encounters isolated integrals which he needs to tackle. Here he typically wants closed form solution, but often settles for an approximate answer. The need here is based on the motivation to "do something with" the result, that is, to either study its dependency on some parameters or on some other mathematical operations. The second context revolves around multiple integration. Here the goal is usually numerical evaluation, but one is interested in reducing the dimensionality of integration as much as possible, because multiple quadrature is costly both in computing time and accuracy. The third context concerns multi-parameter studies, where the integral depends on a number of parameters, thus making numerical results difficult, if not impossible to interprete. For example, if the integral is a function of six parameters the numerical result would require a six-dimensional table or six-dimensional hypersurface to represent. In all these contexts of applications, current technology forces an investigator to take either alternative of the two extremes of numerical versus analytic results (with some exceptions to be mentioned later). It is fair to say that most "real life" problems are non-elegant in nature and for which analytic results are difficult and unlikely to come by. For example, a polynomial of 5th degree whose coefficients are derived from data or other computations are usually irreducible over the integers. In most non-trivial algorithms of integration this fundamental limitation is often fatal, because they involve, in one form or another, partial fraction decomposition which depends on factorization. All these discussions point to the need of a compromising approach between the extremes of numerical and exact integration. Such an approach (let us call it approximate integration is resorted to by scientists and engineers in isolated instances, but has not been investigated as a possible general purpose tool in the sense of a quadrature scheme or a symbolic integration algorithm. The important point to stress is that the approximate approach is intended to yield an output that is an expression, rather than a table of numbers.

At this stage we have examined four broad categories of such approximate schemes. The first consists of the approximation of the integrand by a set of basis functions such as polynomials or splines. There have been some isolated applications using such approximation, for instance, in finite element analysis. One example is given in the particular integration of mass and stiffness matrices given in (Ref. 5). Here the integrand, after a sequence of symbolic manipulations, is made up of a matrix of bivariate polynomials which are readily integrated. In a more general vein, Andersen (Ref. 6) describes the variety of integrations for triangular and quadrilateral finite elements.

The second approach may be labelled interpolatory scheme. Here the spirit is analogous to the derivation of quadrature schemes. i.e., by approximating the integrand by some interpolation formula and then integrating term by term. An example can be cited from Filon quadrature (Ref. 7). Here the integrand is of the form f(x)sico(ax) where sico is either sine or cosine. The integration interval is subdivided into n segments and f(x) is interpolated by a quadratic in each segment to fit the midpoint and two endpoints of that segment. The interpolated expression can then be integrated analytically. Similar techniques

# AUTHOR INDEX

Abdali . . . . . . . . . . . . . 253 3/F10

Ament . . . . . . . . . . . . . 87 2/A7

Andersen . . . . . . . . . . . . 161 2/F11

Anderson . . . . . . . . . . . . 395 5/C8

Avgoustis . . . . . . . . . . . 21 1/C6

Bogen . . . . . . . . . . . 11, 75 1/B10

Bulnes . . . . . . . . . . . . . 461 6/A9

Caviness . . . . . . . . . . . . 253 3/F10

Char . . . . . . . . . . . . . . 53 1/E10

Cohen . . . . . . . . . . . . . 275 4/A9

Cuthill . . . . . . . . . . . . 131 2/E3

Doohovskoy . . . . . . . . . . . 473 6/B7

Fateman . . . . . . . . . . 43, 327 1/D14

Fennelly . . . . . . . . . . . . 97 2/B3

Geddes . . . . . . . . . . . . . 405 5/D4

Genesereth . . . . . . . . . 291, 309 4/B11

Golden . . . . . . . . . . . . 1, 109 1/A14

Gosper . . . . . . . . . . . . . 237 3/E8

Grünbaum . . . . . . . . . . . . 491 6/C11

Gupta . . . . . . . . . . . . . 151 2/F1

Ivie . . . . . . . . . . . . . . 317 4/D9

Karney . . . . . . . . . . . . . . 377 5/B4

Kulp . . . . . . . . . . . . . . . 385 5/B12

Lafferty . . . . . . . . . . . . . 347 4/F11

Lau . . . . . . . . . . . . . . . 395 5/C8

Lewis . . . . . . . . . . . . . . 277 4/A11

Moenck . . . . . . . . . . . . . . 225 3/D10

Moses . . . . . . . . . . . . 123, 275 2/D1

Ng . . . . . . . . . . . . . . 151, 177 2/F1

Noor . . . . . . . . . . . . . . . 161 2/F11

Pavelle . . . . . . . . . . . . 75, 97 1/G4

Poole . . . . . . . . . . . . . . 145 2/E9

Pridor . . . . . . . . . . . . . . 253 3/F10

Rothstein . . . . . . . . . . . . 263 3/G6

Spear . . . . . . . . . . . . . . 369 5/A10

Steele . . . . . . . . . . . . 203, 215 3/C2

Stoutemyer . . . . . . . . 315, 425, 447 4/D7

Wahlquist . . . . . . . . . . . . . 71 1/F14

Wang . . . . . . . . . . . . . 55, 435 1/E12

Yun . . . . . . . . . . . . . . . . 65 1/F8

Zippel . . . . . . . . . . . . . . 361 4/G11

# CONTENTS

FOREWORD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iii 1/A10

PREFACE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . v 1/A12

AUTHOR INDEX . . . . . . . . . . . . . . . . . . . . . . . . . . . vii 1/A1

1. MACSYMA'S SYMBOLIC ORDINARY DIFFERENTIAL EQUATION SOLVER . . . . . . . 1 1/A14
   Jeffrey P. Golden

2. A PROGRAM FOR THE SOLUTION OF INTEGRAL EQUATIONS . . . . . . . . . . . 11 1/B10
   Richard A. Bogen

3. SYMBOLIC LAPLACE TRANSFORMS OF SPECIAL FUNCTIONS . . . . . . . . . . . 21 1/C6
   Yannis Avgoustis

4. AN IMPROVED ALGORITHM FOR THE ISOLATION OF POLYNOMIAL REAL ZEROS . . . 43 1/D14
   Richard J. Fateman

5. FLOATING POINT ISOLATION OF ZEROS OF A REAL POLYNOMIAL VIA MACSYMA . . 53 1/E10
   Bruce W. Char

6. PRESERVING SPARSENESS IN MULTIVARIATE POLYNOMIAL FACTORIZATION . . . . 55 1/E12
   Paul S. Wang

7. ON THE EQUIVALENCE OF POLYNOMIAL GCD AND SQUAREFREE FACTORIZATION PROBLEMS . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65 1/F8
   David Y. Y. Yun

8. DIFFERENTIAL FORM ANALYSIS USING MACSYMA . . . . . . . . . . . . . . . 71 1/F14
   Hugo D. Wahlquist

9. INDICIAL TENSOR MANIPULATION ON MACSYMA . . . . . . . . . . . . . . . 75 1/G4
   Richard A. Bogen and Richard Pavelle

10. PURE FIELD THEORIES AND MACSYMA ALGORITHMS . . . . . . . . . . . . . 87 2/A7
    William S. Ament

11. BLACK HOLES AND RELATIVISTIC GRAVITY THEORIES . . . . . . . . . . . . 97 2/B3
    A. J. Fennelly and Richard Pavelle

12. THE EVALUATION OF ATOMIC VARIABLES IN MACSYMA . . . . . . . . . . . . 109 2/C1
    Jeffrey P. Golden

13. THE VARIETY OF VARIABLES IN MATHEMATICAL EXPRESSIONS . . . . . . . . 123 2/D1
Joel Moses

14. RATIONAL APPROXIMATION TO $e^{-x}$ WITH NEGATIVE REAL POLES . . . . . . . . 131 2/D9
Elizabeth Cuthill

15. TIMING FORMULAS FOR DISSECTION ALGORITHMS ON VECTOR COMPUTERS . . . . 145 2/E9
W. G. Poole, Jr.

16. SYMBOLIC CALCULATIONS IN A FINITE DYNAMIC ELEMENT ANALYSIS . . . . . 151 2/F1
Kajal K. Gupta and Edward W. Ng

17. SYMBOLIC MANIPULATION TECHNIQUES FOR VIBRATION ANALYSIS OF LAMINATED ELLIPTIC PLATES . . . . . . . . . . . . . . . . . . . . 161 2/F11
C. M. Andersen and Ahmed K. Noor

18. OBSERVATIONS ON APPROXIMATE INTEGRATIONS . . . . . . . . . . . . . . . 177 2/G13
Edward W. Ng

19. LISP: PROGRAM IS DATA – A HISTORICAL PERSPECTIVE ON MACLISP . . . . 181 3/A8
Jon L White

20. LISP: DATA IS PROGRAM – A TUTORIAL IN LISP . . . . . . . . . . . . . 191 3/B4
Jon L White

21. DATA REPRESENTATIONS IN PDP-10 MACLISP . . . . . . . . . . . . . . . . 203 3/C2
Guy Lewis Steele Jr.

22. FAST ARITHMETIC IN MACLISP . . . . . . . . . . . . . . . . . . . . . . 215 3/C14
Guy Lewis Steele Jr.

23. ON COMPUTING CLOSED FORMS FOR SUMMATIONS . . . . . . . . . . . . . . 225 3/D10
Robert Moenck

24. INDEFINITE HYPERGEOMETRIC SUMS IN MACSYMA . . . . . . . . . . . . . . 237 3/E8
R. Wm. Gosper, Jr.

25. MODULAR POLYNOMIAL ARITHMETIC IN PARTIAL FRACTION DECOMPOSITION . . . 253 3/F10
S. K. Abdali, B. F. Caviness, and A. Pridor

26. A NEW ALGORITHM FOR THE INTEGRATION OF EXPONENTIAL AND LOGARITHMIC FUNCTIONS . . . . . . . . . . . . . . . . . . . . . . . 263 3/G6
Michael Rothstein

27. SUMMATION OF RATIONAL EXPONENTIAL EXPRESSIONS IN CLOSED FORM . . . . 275 4/A9
Joel Moses and Jacques Cohen

28. USER AIDS FOR MACSYMA . . . . . . . . . . . . . . . . . . . . . . . . 277 4/A11
V. Ellen Lewis

29. THE DIFFICULTIES OF USING MACSYMA AND THE FUNCTION OF USER AIDS . . . 291 4/B11
Michael R. Genesereth

30. AN AUTOMATED CONSULTANT FOR MACSYMA . . . . . . . . . . . . . . . . . . 309 4/D1
Michael R. Genesereth

31. A MACSYMA COMPUTER-ALGEBRA MOVIE DEMONSTRATION . . . . . . . . . . . . 315 4/D7
David R. Stoutemyer

32. SOME MACSYMA PROGRAMS FOR SOLVING DIFFERENCE EQUATIONS . . . . . . . 317 4/D9
John Ivie

33. SOME COMMENTS ON SERIES SOLUTIONS . . . . . . . . . . . . . . . . . . 327 4/E5
Richard J. Fateman

34. POWER SERIES SOLUTIONS OF ORDINARY DIFFERENTIAL EQUATIONS IN MACSYMA . . . . . . . . . . . . . . . . . . . . . . . . . . 347 4/F11
Edward L. Lafferty

35. RADICAL SIMPLIFICATION MADE EASY . . . . . . . . . . . . . . . . . . . 361 4/G11
Richard E. B. Zippel

36. A CONSTRUCTIVE APPROACH TO COMMUTATIVE RING THEORY . . . . . . . . . 369 5/A10
David A. Spear

37. REDUCTION OF THE EQUATION FOR LOWER HYBRID WAVES IN A PLASMA TO A NONLINEAR SCHRÖDINGER EQUATION . . . . . . . . . . . . . . . . 377 5/B4
Charles F. F. Karney

38. RAY TRAJECTORIES IN A TORUS: AN APPLICATION OF MACSYMA TO A COMPLEX NUMERICAL COMPUTATION . . . . . . . . . . . . . . . . . . . . . . 385 5/B12
John L. Kulp

39. APPLICATION OF MACSYMA TO FIRST ORDER PERTURBATION THEORY IN CELESTIAL MECHANICS . . . . . . . . . . . . . . . . . . . . . . . . 395 5/C8
John D. Anderson and Eunice L. Lau

40. SYMBOLIC COMPUTATION OF RECURRENCE EQUATIONS FOR THE CHEBYSHEV SERIES SOLUTION OF LINEAR ODE's . . . . . . . . . . . . . . . . . 405 5/D4
K. O. Geddes

41. sin(x)**2 + cos(x)**2 = 1 . . . . . . . . . . . . . . . . . . . . . . 425 5/E10
David R. Stoutemyer

42. MATRIX COMPUTATIONS IN MACSYMA . . . . . . . . . . . . . . . . . . . . 435 5/F6
Paul S. Wang

43. SYMBOLIC COMPUTER VECTOR ANALYSIS . . . . . . . . . . . . . . . . . . 447 **5/G4**
David R. Stoutemyer

44. A NATURAL WAY TO DO SPATIAL LINEAR GEOMETRY IN MACSYMA . . . . . . . 461 **6/A9**
Juan Bulnes

45. VARIETIES OF OPERATOR MANIPULATION . . . . . . . . . . . . . . . . . 473 **6/B7**
Alexander Doohovskoy

46. PROGRESS REPORT ON THE DETERMINANT OF A RANDOM MATRIX . . . . . . . . 491 **6/C11**
F. A. Grünbaum

BIBLIOGRAPHY . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 493 **6/C13**

can be applied to other types of functions. As pointed out by a referee, interpolation actually can be viewed as a special case of approximating in terms of a basis, it being the Lagrange polynomials associated with the interpolation points and having an integral error criterion subject to exact fit at these points.

The third approach is based on a reduction of transcendence of the integrand. Termwise integration of approximations of the integrand by power or asymptotic series is a well-known example in this category. This strategy amounts to an approximation of the integrand by a polynomial. However, one can also approximate the integrand by a rational function. For example, take the exponential of a polynomial. For a proper range the exponential can be approximated by a rational function, but there is an associated difficulty here, namely, that the rational function consists of polynomials of high degrees, and that some kind of telescoping procedure need be applied in order that the integrated result is manageable. An example will be presented to detail the advantages and disadvantages of such a strategy.

The last approach is to compute the integral by quadrature and then approximate the answer by, for example, some basis functions. This approach can hardly be considered under the umbrella of integration (it is more of a curve or surface fitting problem). In a paper on practical approximations (Ref. 8) the author gives an example on the approximation of an integral. The basic idea will carry through to a more general problem where quadrature can be used instead. We shall comment on the pros and cons of this approach.

In the oral presentation we shall provide a concrete example for each approach and discuss the MACSYMA relevance to each. Though we do not have a coherent theory behind each, we believe this investigation is a modest beginning of approaches of practical significance.

## REFERENCES

1. Risch, R.: The Problem of Integration in Finite Terms. Trans. American Math. Soc., vol. 139, 1969, pp. 167-189.

2. Moses, J.: Toward A General Theory of Special Functions. Commun. ACM, vol. 15, no. 7, July 1972, pp. 550-554.

3. Wang, P.: Evaluation of Definite Integrals by Symbolic Manipulation. Rep. 92, Lab. Comput. Sci., (formerly Proj. MAC), Massachusetts Inst. Technol., 1971.

4. Ng, E.; and Polajnar, D.: A Study of Alternative Methods for Symbolic Calculation of Elliptical Integrals. Proceeding of 1976 ACM Symposium on Symbolic and Algebraic Computations, Aug. 1976, pp. 372-376.

5. Gupta, K.; and Ng, E.: Symbolic Calculations in a Finite Dynamic Element Analysis. Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper no. 16 of this compilation.)

6. Andersen, C.: Use of Computerized Symbolic Integration in Finite Element Development. Proceedings of the ACM Annual Conference, pp. 554-562, 1974.

7. Chase, S.M.; and Fosdick, L.D.: An Algorithm for Filon Quadrature. Commun. ACM, vol. 12, no. 8, Aug. 1969.

8. Cody, W.J.: A Survey of Practical Rational and Polynomial Approximation of Functions. SIAM Rev. vol. 12, no. 3, 1970.

# LISP: PROGRAM IS DATA

## A HISTORICAL PERSPECTIVE ON MACLISP

Jon L White
Laboratory for Computer Science, M.I.T.*

## ABSTRACT

For over 10 years, MACLISP has supported a variety of projects at M.I.T.'s Artificial Intelligence Laboratory, and the Laboratory for Computer Science (formerly Project MAC). During this time, there has been a continuing development of the MACLISP system, spurred in great measure by the needs of MACSYMA development. Herein are reported, in a mosiac, historical style, the major features of the system. For each feature discussed, an attempt will be made to mention the year of initial development, and the names of persons or projects primarily responsible for requiring, needing, or suggesting such features.

## INTRODUCTION

In 1964, Greenblatt and others participated in the check-out phase of Digital Equipment Corporation's new computer, the PDP-6. This machine had a number of innovative features that were thought to be ideal for the development of a list processing system, and thus it was very appropriate that the first working program actually run on the PDP-6 was an ancestor of the current MACLISP. This early LISP was patterned after the existing PDP-1 LISP (see reference 1), and was produced by using the text editor and a mini-assembler on the PDP-1. That first PDP-6 finally found its way into M.I.T.'s Project MAC for use by the Artificial Intelligence group (the A.I. group later became the M.I.T. Artificial Intelligence Laboratory, and Project MAC became the Laboratory for Computer Science). By 1968, the PDP-6 was running the Incompatible Time-Sharing system, and was soon supplanted by the PDP-10. Today, the KL-10, an advanced version of the PDP-10, supports a variety of time sharing systems, most of which are capable of running a MACLISP.

MACSYMA (ref. 2) grew out of projects started on the 7090 LISP 1.5, namely Moses' SIN program and Martin's MATHLAB. By implementing the Project MAC Symbolic and Algebraic manipulation system in LISP, many advantages were obtained. Of particular importance were (i) a basic data convention well-suited for encoding algebraic expressions, (ii) the ability for many independent individuals to make programming contributions by adhering to the programming and data framework of LISP, and (iii) the availability of a good compiler and debugging aids in the MACLISP system. As the years rolled by, the question was asked "What price LISP"? That is, how much faster could the algebraic system be if the advantages brought by the LISP system were abandoned and an all-out effort was made in machine language? Moses has estimated that about a factor of two could be gained (private communication), but at the cost of shifting much of the project resources from mathematical research to coding and programming. However, that loss could have been much larger had not MACLISP development kept pace, being inspired by the problems observed during MACSYMA development, and the development of other projects in the A.I. Laboratory. The most precarious strain placed on the supporting LISP system by MACSYMA has been its sheer size, and this has led to new and fundamental changes to MACLISP, with more yet still in the future. Many times, the MACSYMA system was not able to utilize the solution generated for one of its problems, due to the familiar trap of having already too much code invested in some bypass solution; but there has generally been an interchange of ideas amongst those groups using MACLISP at the A.I. Lab and LCS, and another group may have received the benefit of an idea born by MACSYMA needs.

---

*During the calendar year 1977, the author is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

Because the system is still evolving after a decade of development, it is useful to think of it as one big piece of *data*, a *program* still amenable to further critical review and emendation. Below are presented some of the developments of this past 10 years, with a little bit of explanation as to their significance and origin.

## HOW WE GOT TO WHERE WE ARE

### Clever Control Features

In 1966, Greenblatt suggested abandoning the a-list model for program variables, and returning to a standard save-and-restore stack model such as might be used by a recursive FORTRAN. This was the first LISP to do so, and a later LISP developed at Bolt, Beranek, and Newman (BBN) in Cambridge used a model whereby storage for program variables was dynamically allocated on the top of a stack. Both stack models could achieve a significant speed-up over the a-list models, but at a cost of limiting the use of FUNCTION (see ref. 3). The BBN LISP later became INTERLISP (ref. 4), and currently has a stack model with the same function capabilities as the a-list model. In 1975, the PROGV feature was added and is apparently unique to MACLISP. PROGV is essentially PROG, except that the list of variables is not syntactically present, but rather is computed as an argument to PROGV; previously, about the best one could do was to call EVAL (or APPLY) with a dynamically-constructed LAMBDA expression.

In 1969, Sussman, noticing features of the MULTICS operating system, demanded some similar features for MACLISP: asynchronous interruption capability, such as alarmclocks, job-console control keys, hardware faults, interprocess communication, and exceptional process conditions (chiefly, errors). Many LISP systems now permit the user to supply functions for handling standard LISP errors, and provide for some mechanism at the job-console to interrupt the system, putting it into a top-level-like loop called BREAK. MACLISP permits interruption capability on any character of the input-console keyboard; the user may designate any function to be run when a particular key is typed. To some degree, these features appeared concurrently in INTERLISP, but especially the stackframe and debugging facilities of INTERLISP inspired similar ones in MACLISP. In mid-1976, MACLISP could finally give an interrupt to the user program on several classes of hardware-detected conditions: access (read or write) to a specific address, attempted access to non-existent address, attempted write access into read-only memory, parity error, and illegal instruction. Furthermore, some operating system conditions could trigger special interrupts: system about to shut down in a few minutes, and console screen altered by system. Evident from the development of LISP-embedded systems was the need for a NOINTERRUPT facility, which could protect user-coded processes from an accidental, mid-function aborting such as might occur during an asynchronous interrupt. Steele designed and implemented the current scheme in late 1973.

Sussman's development of MICRO-PLANNER (ref. 5) required some more capabilities for intelligent, dynamic memory management; and thus White, in 1971, introduced programmable parameters for the garbage collector — a minimum size for each space, a maximum allowable, and a figure demanding that a certain amount be reclaimed (or found free) after a collection. Then in the next year came the GC-DAEMON mechanism, whereby a user function is called immediately after each garbage collection so that it can intelligently monitor the usage of memory and purposefully modify the memory-management parameters. Baker, who has recently done work on concurrent garbage collection (ref. 6), has produced a typical storage monitor using the MACLISP mechanisms (ref. 7).

Sussman's later development of CONNIVER (ref. 8) showed the need for a sort of non-local GOTO, as a means of quickly aborting a computation (such as a pattern-matching data-base search) that had gone down a wrong path. Thus in 1972 White devised the CATCH and THROW facilities (THROW provides a quick, non-local break-out to a program spot determined by CATCH), and implemented FRETURN as a means of an impromptu "THROW" out of any stackframe higher up than the current point of computation (this is especially effective if an error break occurs, and the user can supply by hand a correct return value for some pending subroutine call several levels up the stack). In 1975, Steele coded the EVALHOOK feature, which traps each interpretive entry to EVAL during the evaluation of a piece of code; this permitted users to write debugging packages that can effectively "single-step" through an evaluation.

The embedding of advanced programming-language systems in LISP, such as MACSYMA, MICRO-PLANNER, CONNIVER, and LLOGO (ref. 9) required a means of insulating the supporting system (written as LISP code) from the users code (written in the new experimental language). Sussman and White noticed that the action of INTERN was primarily a table look-up, and they implemented this table (in 1971) as a LISP array, which array is held as the value of the global variable OBARRAY. Thus a user can change, or even LAMBDA-bind, the INTERN environment. Similarly, the action of the programmable reader could be controlled by exposing its syntax and macro table as the value of the global variable READTABLE, which was done in 1972. In 1975, the MAPATOMS function as found in INTERLISP was implemented for quickly applying a function to all the objects on a given OBARRAY. All these embedded systems wanted to have better control over the LISP top-level and break-level loops; so in 1971 two features were added: 1) ability to replace the top-level and break-level action with a form of the user's choice, and 2) a facility to capture control after a system-detected error has occurred but before re-entry to the top level. At first, the error-break permitted only exiting by quitting out back to top level, but later these breaks were such that many errors could be corrected and the computation restarted at the point just prior to the error detection. By early 1975, it was noted that many applications wanted to alter what might be called the default input reader and the default output printer; the former because their code files were written with many macro and special facilities, and the latter because of the occurrence of circular list structure. Thus the two variables READ and PRIN1, if non-NIL, hold a user-supplied function for these operations.

### I/O Facilities

In 1968, White proposed a programmable, macro-character input reader, and by the summer of 1969, the reader was in operation. Since that time, some other LISPs have added certain special features to their readers, such as inputting 'A as (QUOTE A), or as in INTERLISP, permitting the user to change the meaning of break, separator, and escape characters; but to the author's knowledge none have any user-programmable macro[1] facility, nor so wide a range of parsing options as does MACLISP.

The PRINT function of MACLISP has remained relatively neglected over the years; but in 1973 Steele implemented the PRINLEVEL and PRINLENGTH facilities as inspired by the INTERLISP PRINTLEVEL facility. LISP has always had the notion of "line length", such that if more than a specified number of characters were output without an intervening newline character, the a newline was automatically inserted by the system (this was especially practical in the days when model 33 Teletypes were the main terminal used, and the operating system did not take care of preventing too long a line). MACLISP allowed an override on this automatic insertion feature, but in 1976 Steele modified this facility so that, even when not overriden, it would not insert the generated newline character in the middle of some atom. Along with the macro-reader in 1968, White installed dynamically-variable base conversion for fixnums, so that any base between 2 and 36 could be used; for what it's worth, Steele extended this for roman numerals also in 1974.

---

[1] Of course the macro functions are written in LISP, what else!

The problem of "perfect" output for floating-point numbers on the PDP-10 has apparently not been solved in any other system. That is, given the more-or-less standard input algorithm for base conversion from floating-point decimal numbers (dfpns) to floating-point binary numbers (bfpns), construct an output conversion algorithm such that

i) every representable bfpn is converted to a shortest dfpn, and

ii) if e is a representable bfpn, and $e^*$ is its dfpn image by the output algorithm, then the input algorithm applied to $e^*$ produces *exactly* e.

In 1972, White devised and installed in MACLISP an algorithm that was more nearly "perfect" than any other known to the author or to persons of his acquaintance; and in May 1977 White and Steele improved that algorithm so that they think it is "perfect" (a proof of which is forthcoming). Most other algorithms will increase the least-significant bit of some numbers when passed through the read-in of print-out cycle (see reference 10 for a possible explanation of why this problem is so hard). Golden anticipates MACSYMA's usage of this capability, "perfect" print-out, if it indeed is truly so.

Inspired by LISP 1.6 (ref. 11), a preliminary version of a multiple I/O scheme was coded up by Stallman in 1971. Prior to this, MACLISP could effectively READ from at most one file at a time, and PRINT out onto at most one file at a time; furthermore, there were no provisions for I/O other than the ASCII streams implicit in READ and PRINT. That preliminary version was abandoned in early 1973, and a decision was made to copy the design of the MULTICS version I/O (which had been developed rather independently). This scheme, coded by Steele and ready for use early in 1975, has been termed "Newio". It has since been undergoing continuing check-out and development up until now, and in January 1977 became the standard MACLISP on the ITS versions, although we have not yet made the necessary modifications to the TOPS-10 version.

Between 1967 and 1971, the A.I. Lab Vision Group, and MACSYMA Group saw the need for a faster method of getting compiled LISP subroutines off disk storage and into a running system. Back then, the compiler would produce a file of LAP code, which would be assembled in each time it was required. The first step in this direction was taken in 1969 when White devised a dynamic array space, with automatic garbage collection. Then White and others worked out a relocatable format for disk storage such that the load in time could be minimal; Steele and White implemented this scheme between 1972 and 1973, called FASLOAD. Golden reported that the time to load in all the routines comprising the then-existing MACSYMA dropped from about an hour to two minutes; continuing MACSYMA development certainly required this FASt LOADing scheme. Closely following in time was the AUTOLOAD scheme, whereby a function that was not part of the in-core environment, but resident in FASL format on disk, would be FASLOADed in upon first invocation.

## Arithmetic Capabilities

Perhaps the most stunning achievement of MACLISP has been the method of arithmetic that has permitted FORTRAN-like speed from compiled LISP code. In 1968, Martin and Moses, foreseeing future needs of MACSYMA, demanded better arithmetic capabilities from MACLISP. In 1969, Martin changed the implementation of numbers so that FIXNUMs and FLONUMS consumed only one word, rather than three — that is, the LISP 1.5 format was abandoned and numbers were implemented merely as the pointer to the full-word space cell containing their value. Such a scheme had already been accomplished, partially, in other LISPs. After that change in the interpreter had been completed, some new functions were introduced for type-specific arithmetic:

for fixed point: + - * / 1+ 1-

for floating point: +$ -$ *$ /$ 1+$ 1-$

for either (but not mixed): = < >

Later, more functions were added, such as fixed-point square-root, and greatest-common-divisor. The fixed-point functions would be an automatic declaration to the compiler that all arguments and results would be fixnums, and that all arithmetic can be modulo $2^{35}$; similarly, the flonum functions would specify the use of floating point hardware in the compiled code.

At the same time, Binford suggested installing separate full-word stacks for FIXNUMs and for FLONUMs, and interpreting these stack addresses as the corresponding type number. Then White proposed eliminating the discontinuity in FIXNUM representation caused by the INUM scheme, so that open-compilation of numeric code would need no extra, interpretive-like steps to extract the numerical value from a LISP number.[2] White also designed a scheme for using the number stacks, interfacing compiled subroutines with one another and with the interpreter. The redesign of number storage, and the design of a numeric subroutine interface, was for the purpose of permitting the compiler to produce code similar to what a PDP-10 FORTRAN compiler could produce on essentially numeric programs.[3] Work then began on the compiler to take advantage of all this, and a preliminary version for arithmetic code was operational by late 1971, under the care of Golden and Rosen who did most of the early coding. Rosen and White developed optimization in the compiler during 1972, and White continued this work through the end of 1976. In 1974, White and Steele extended the array data facilities of MACLISP to include FORTRAN-like arrays of fixnums and flonums so that the compiler could optimize array references in numerical code; see Steele's paper describing the current output available from the compiler (ref. 13).

Early along in MACSYMA development, Moses and Martin saw the need for variable-precision integer arithmetic, and thus the BIGNUM functions were born, with most algorithms taken from Knuth (ref. 14). During 1972 and 1973, Golden suggested the need in MACSYMA for some of the usual transcendental functions, like SIN, COS, natural logarithm and anti-logarithm, and arc-tangent (these were adapted from some rational approximations originally developed by White in 1967); for GCD, HAULONG, HAIPART, and improvements to the the exponentiation function EXPT; and for the ZUNDERFLOW switch, which permits interpretive arithmetic routines to substitute a real zero for any floating-point result that causes a floating-point underflow condition. By combining the binary and Lehmer algorithms from Knuth (ref. 15), Gosper produced a GCD algorithm early in 1976 which runs much faster on bignum inputs. Also, in 1976, a feature was added to the interpretive floating-point addition and subtraction routines such that if the sum is significantly less than the principal summand, then the sum is converted to zero; the variable ZFUZZ holds a scale-factor for this feature, which is still considered experimental (LISP370 has a more pervasive use of a similar feature in all floating-point arithmetic and I/O functions).

Randomness has always been a property of MACLISP, having had a linear-shift-register RANDOM number generator since early times. This generator produced a maximally-long sequence, was extremely fast, and moderately acceptable for most applications. However, it failed the correlated-triples test, and when it was used to generate random scenes for display on the LOGO Advent color projector, it produced some very nice kaleidoscopic pictures; so in late 1976, a modification of Knuth's Algorithm A (ref. 16) was coded by Horn.

### Ancillary Packages

A number of ancillary functions have been coded in LISP, mostly by persons who were LISP users rather than system developers, and are kept stored in their compiled, FASL format for loading in when desired. In 1970, Binford coded a small, but powerful, subset of the INTERLISP in-core editor as a LISP package, but this was later recoded in machine language; a more extensive version of the INTERLISP editor has been coded by Gabriel in 1975. In 1970, Winston designed and coded INDEX, a package to analyze a file of LISP programs and report on certain properties therein. During 1972, Goldstein replaced an existing, slow pretty-printer (called GRIND) with a programmable pretty-printer (ref. 17), and Steele spruced-up an existing TRACE package to have more features. After the Newio scheme became operational, two packages were coded for the fast dumping onto disk and retrieval therefrom of numeric arrays, and a FASDUMP package was implemented for MACSYMA that could quickly and efficiently store list structure on disk (Kulp had a hand in developing this package, but it may no longer be in use). Many of these user-supplied packages now reside on a disk area called LIBLSP, which includes a FORMAT package by White for printing out numbers under control of a format (such as is used in FORTRAN), a package for reading and printing circular list structures, various debugging packages and s-expression editors, and many others.

---

[2] MACLISP, by inspecting the numerical value of a number coming into the FIXNUM-conser, supplies a canonical, read-only copy for fixnums in the range of about -1000. to +2000. This significantly reduces the number of new cells required by running arithmetic code, without significantly slowing down the operations. Currently, no similar action is taken for FLONUMs.

[3] The generally-accepted opinion in 1968, and indeed in some quarters up until 1973, was that LISP is *inherently* a hundred times slower on arithmetic than is FORTRAN. Fateman's note in 1973 effectively rebutted this opinion (ref. 12), but in 1969 it took faith to go ahead with this plan; only Martin and the author had a clear resolve to do so then.

In 1973 Pratt was continuing work on a "front end" for LISP, CGOL (ref. 18), which he had begun at Stanford University in 1971, and he had it generally operational at a number of sites by 1975. It exemplifies the Pratt operator-precedence parser (now used at the front end of MACSYMA), and has some of the character of MLISP (ref. 19). However, the CGOL-to-MACLISP conversion is dynamic and fast, and furthermore, an acceptable inverse operation has been implemented, so that one can effectively use this ALGOL-like language while still retaining all the advantages of MACLISP (fast interpreter, good compiler, many debugging aids, etc.). It is not at all impractical to replace the MACLISP default reader and printer with CGOL's (see notes on READ and PRIN1 in the last paragraph of "Clever Control Features" above), so that CGOL may be properly thought of as an alternate external syntax for LISP. See reference 7 for a practical example — one particular GC-DAEMON function for MACLISP, coded in CGOL.

MIDAS, the A.I. Lab's assembly-language system for the PDP-10, cooperates with MACLISP to the extent of being able to produce a FASL format file. A number of these ancillary packages have thus been coded in machine language for greater efficiency. In mid 1973, Steele coded a version of Quicksort (ref. 20) which is autoloadable as the function SORT; in 1976, after Newio became stable, Steele coded a file-directory query package (called ALLFILES), and designed a package for creating and controlling subjobs (tasks) in the ITS time-sharing environment (called HUMBLE). Using the HUMBLE package, Kulp and others interfaced the text editor TECO with MACLISP, for increased programmer efficiency in debugging and updating LISP programs. Kulp and others had proposed a text-processing system suitable for use with a photo-composer to be written in MACLISP and using these features, but this has not yet been realized. With the ALARMCLOCK facility for periodic interrupts, and HUMBLE for driving sub-tasks, MACLISP is fully equipped for becoming a time-sharing system.

## Export Systems

Martin's desire to be able to use MACSYMA on the MULTICS system led to the start of a MULTICS version of MACLISP, begun in late 1971 by Reed; after this was fully operational in 1973, Moon, who had worked on it wrote the now-extinct MACLISP Reference Manual published in March 1974 (ref. 21). Although there has been little use of MACSYMA on the MULTICS version, it was successfully transplanted there; several other extension systems developed on the PDP-10 version were also successfully tested on the MULTICS version, such as LLOGO and CONNIVER.

In the summer of 1973, the MACLISP system was extended to permit its use on TOPS-10, DEC's non-paged time sharing system. Much help on this development has come from members of the Worcester Polytech Computation Center, and from the resources of the Computer Science department of Carnegie-Mellon University. The impetus for having a TOPS-10 version came from many academic institutions, where students with interests in artificial intelligence had been intrigued by MICRO-PLANNER and CONNIVER and their applications, and had wanted to experiment with these systems on their own PDP-10s. Later, as M.I.T. graduate students and professors moved to other universities, they took with them the desire to use MACLISP, rather than any of the other available LISP alternatives. The major difficulty in export to these other institutions has been their lack of adequate amounts of main memory — few places could even run the MACLISP compiler, which requires 65*K. At one time Moses had a desire to export MACSYMA through this means, but this has not proved feasible. Even for the KI-10 and KL-10 processors, which have paging boxes, the TOPS-10 operating system does not give user programs sufficient control over the page-map; consequently, this version of MACLISP is to some degree less efficient in its memory utilization.

The TENEX and TOPS-20 operating systems should be able to support the TOPS-10 version of MACLISP, under a compatibility mode, but there has been some difficulty there. In 1971, a specially tailored version of MACLISP was run under the TENEX system, but this version died out for lack of interest. If future interest demands it, there should be no trouble in getting almost the full range of MACLISP features found on the ITS version to be implemented in a TOPS-20/TENEX version. In 1976 Gabriel adapted the TOPS-10 version to run on the Stanford A.I. Laboratory operating system, and there is currently an increasing body of users out there.

## Revised Data Representations

A major step was taken in 1973 when the long-awaited plans to revise the storage strategy of MACLISP saw the light. A plan called Bibop (acronym for BIg Bag Of Pages), inspired in part by the prior INTERLISP format, was designed by White, Steele, and Macrakis; and this was coded by Steele during the succeeding year. The new format relieves the need for a LISP user to make precise allocations of computer memory, and permits dynamic expansion of each data space (although only the array storage area can be dynamically reduced in size). In 1974, numeric arrays were added, and in 1976 a new data type called HUNK was added as a s-expression vector without any of the overhead associated with the array data type. Steele's paper in these proceedings (ref. 22) gives a detailed account of how the current storage picture looks inside MACLISP.

Especially MACSYMA, as well as Winograd's SHRDLU and Hewitt's PLASMA systems, needed the efficiency and versatility of these new formats. The concept of "pure free storage" entered the picture after Bibop became operational: this is list and s-expression structure that is essentially constant, and which can be removed from the active storage areas that the garbage collector manages. Furthermore, it can be made read-only, and shared among users of the same system; in MACSYMA, there are myriads of such cells, and the consequent savings is enormous. Thus the incremental amount of memory required for another MACSYMA user on the system starts at only about 45K words!

## The Compiler

Greenblatt and others wrote a compiler for the PDP-6 lisp, patterned initially after the one for 7090 LISP on CTSS. This early attempt is the grandfather of both the current MACLISP and current LISP 1.6 compilers. However, optimizing LISP code for the the PDP-6 (and PDP-10) is a much more difficult task than it might first appear to be, because of the multiple opportunities provided by the machine architecture. That early compiler had too many bugs to be really useful, but it did provide a good, basic structure on which White began in 1969 (joined by Golden in 1970) to work out the plans for the fast-arithmetic schemes (see ref. 13). The LISP 1.6 compiler has apparently not had so thorough a check-out and debugging as the MACLISP compiler, since its reputation is unreliability. The INTERLISP compiler was produced independently, and seems to be quite reliable; but comparisons have shown that average programs compile into almost twice as many instructions through it than through the MACLISP compiler.

## Ad-Hoc Hacks

As the number of new and interactive features grew, there was observed need for a systematic way to query and change the status of various of the operating system and LISP system facilities. We did not want to have to introduce a new LISP primitive function for every such feature (there are scores!), so thus was born in 1969 the STATUS and SSTATUS series. The first argument to these functions selects one of many operations, ranging from getting the time of day from a home-built clock, to reading the phase of the moon, and to setting up a special TV terminal line to monitor the garbage collector. Later, in 1975, the function SYSCALL was added as a LISP entry into the time-sharing system's CALL series of operations. (See reference 23 for information on the ITS system.)

Between 1970 and 1972, the demands of the A.I. Lab Vision group necessitated the installation of a simulated TV camera, called the FAKETV, along with a library file of disk-stored scene images. A cooperative effort between the Vision group and the LOGO group led to the design of a Display-slave — a higher, display-orientated language for use with the Lab's 340 Display unit using the PDP-6 as an off-line display processor. Goldstein, because of his interest in LLOGO (ref. 9), participated in the initial design along with Lerman and White; the programming and coding were done by the latter two.

In 1973, terminal-input echo processing (rubout capability) was enhanced, and cursor control was made available to the user for the existing display terminals. When the A.I. Lab began using the home-built TV terminal system, Lieberman coded a general-purpose display packages in LISP for use on the TV display buffer. When Newio became available in 1975, Lieberman and Steele showed examples of split-screen layouts usable from LISP, and in 1976 Steele showed how to code a variety of "rubout" processors in LISP. Furthermore, Newio permitted extended (12-bit) input from the keyboards associated with these terminals.

In 1973, MACLISP copied a feature from LISP 1.6 for improving facilities in linkage between compiled subroutines — the UUOLINKS technique. All compiled subroutine calls are done indirect through a table, which contains interpretive links for subroutine-to-subroutine transfer. Under user option, these links may be "snapped" during run time — that is, converted to a single PDP-10 subroutine transfer instruction. A read-only copy is made of this table (after a system such as MACSYMA is generated) so that it may be restored to its unsnapped state at any time. The advantage of this is that, normally, subroutine transfers will take place in one or two instruction executions, but if it is desired to debug some already compiled subroutines, then one need only restore the interpretive links from the read-only copy.

Inspired by MACSYMA's history variables, MACLISP adopted the convention in early 1971 that the variable "*" would hold the most recent quantity obtained at top level.

In 1973, White coded an s-expression hashing algorithm called SXHASH, which has been useful to routines doing canonicalization of list structure (by hashing, one can greatly speed-up the search to determine whether or not there is an s-expression copy in a table EQUAL to a given s-expression).

To accommodate the group that translated the lunar rocks query-information system from INTERLISP to MACLISP, the convention was established in 1974 that car[NIL]=cdr[NIL]=NIL. This seems to have been widely accepted, since it simplifies many predicates of the form (AND X (CDR X) (CDDR X)) into something like (CDDR X).

## WHERE DO WE GO FROM HERE?

The major problem now with MACLISP, especially as far as MACSYMA is concerned, is the limitation imposed by the PDP-10 architecture — an 18.-bit address space, which after overhead is taken out, only leaves about 180K words for data and compiled programs. Steele discusses some of our current thinking on what to do about this in his paper (ref. 22) of these proceedings, under the section "The Address Space Problem". Since the LISP machine of Greenblatt (ref. 24) is such an attractive alternative, and is even operational now in 1977, we will no doubt explore the possibilities of incorporating into PDP-10 MACLISP some of its unique features, and in general try to reduce the differences between them. For the future of MACSYMA, we foresee the need for new, primitive data types for efficient use of complex numbers and of double-precision floating-point numbers. We anticipate also the need to have a version efficiently planted in the TOPS-20 system.

## FULL NAMES OF PERSONS ASSOCIATED WITH MACLISP DEVELOPMENT AND MENTIONED IN THIS PAPER

**MIT Professors**

Joel Moses
William A. Martin
Gerald J. Sussman
Ira P. Goldstein
Vaughan Pratt
Patrick H. Winston
Terry L. Winograd*
Carl E. Hewitt
Richard J. Fateman*
Berthold K. P. Horn

**Research Staff**

Jon L White
Jeffrey P. Golden
Richard Greenblatt
Thomas O. Binford*
Jerry B. Lerman*
R. William Gosper*

**Students**

Guy L. Steele Jr.
David A. Moon
Eric C. Rosen*
John L. Kulp
Richard P. Gabriel*
Henry Lieberman
Richard M. Stallman
Stavros Macrakis
David P. Reed
Henry G. Baker, Jr.

* = No longer at M.I.T.

## REFERENCES

1. Deutsch, L., and Berkeley, E.; The LISP Implementation for the PDP-1 Computer, in *THE PROGRAMMING LANGUAGE "LISP"*, edited by Berekeley, E., and Bobrow, D., Information International Inc., 1964.
2. *MACSYMA Reference Manual*, Project MAC Mathlab Group, M.I.T., November 1975.
3. Moses, J.; *The Function of FUNCTION in LISP*. AI Memo 199, Artificial Intelligence Lab, M.I.T., June 1970.
4. Teitelman, W.; *INTERLISP Reference Manual (Revised edition)*. Xerox Palo Alto Research Center, 1975.
5. Sussman, G., Winograd, T. and Charniak, E.; *Micro-Planner Reference Manual (revised)*. AI Memo 203A, Artificial Intelligence Lab, M.I.T., December 1971.
6. Baker, H.; A Note on the Optimal Allocation of Spaces in MACLISP. Working Paper 142, Artificial Intelligence Lab, M.I.T., March 1977.
7. Baker, H.; List Processing in Real Time on a Serial Computer. Working Paper 139, Artificial Intelligence Lab, M.I.T., February 1977.
8. McDermott, D., and Sussman, G.; *THE CONNIVER REFERENCE MANUAL*. AI Memo 259A, Artificial Intelligence Lab, M.I.T., January 1974.
9. Goldstein, I.; *LLOGO: An Implementation of LOGO in LISP*. AI Memo 307, Artificial Intelligence Lab, M.I.T., June 1974.
10. Matula, D.; In-and-Out Conversions. *CACM 11, 1*, January 1968, pp. 47-50.
11. Quam, L.; *STANFORD LISP 1.6 MANUAL*. SAILON 28.3, Artificial Intelligence Lab, Stanford University, 1969.
12. Fateman, R.; "Reply to an Editorial", *SIGSAM Bulletin*, 25, March 1973, pp. 9-11.
13. Steele, G.; Fast Arithmetic in MACLISP. Proceedings of the 1977 MACSYMA Users Conference, NASA CP-2012, 1977. (Paper no. 22 of this compilation).
14. Knuth, D.; *The Art of Computer Programming*, V2. Addison-Wesley, 1969, pp. 229-240.
15. ———, *ibid.*, pp. 293-307.
16. ———, *ibid.*, pp. 26-27.
17. Goldstein, I.; *Pretty-Printing, Converting List to Linear Structure*. AI Memo 279, Artificial Intelligence Lab, M.I.T., Feburary 1973.
18. Pratt, V.; CGOL — An Alternative External Representation for LISP Users. Working Paper 121, Artificial Intelligence Lab, M.I.T., March 1976.
19. Smith, D.; *MLISP*. AIM-135, Artificial Intelligence Lab, Stanford University, 1970.
20. Knuth, D.; *The Art of Computer Programming*. V3, Addison-Wesley, 1973, pp. 114-116.
21. Moon, D.; *MACLISP Reference Manual, Revision 0*. Laboratory for Computer Science (formerly Project MAC), M.I.T., March 1974.
22. Steele, G.; Data Representations in PDP-10 MACLISP. Proceedings of the 1977 MACSYMA Users Conference, NASA CP-2012, 1977. (Paper no. 21 of this compilation).
23. Eastlake, D.; *ITS Status Report*. AI Memo 238, Artificial Intelligence Lab, M.I.T., April 1972.
24. Greenblatt, R.; The Lisp Machine. Working Paper 79, Artificial Intelligence Lab, M.I.T., November 1974.

BLANK PAGE

BLANK PAGE

# LISP: DATA IS PROGRAM

## A TUTORIAL IN LISP

Jon L White
Laboratory for Computer Science, M.I.T.*

## ABSTRACT

A novel approach at teaching LISP to a novice is herein developed. First, the abstract data format is presented, emphasizing its real structure and its machine implantation. Then the technique of writing programs in the data language, and of "interpreting" them, is presented. Illustrative features are drawn from various extant LISP implementations.

## INTRODUCTION

The design of LISP as a programming language was based on the desire for a practical implementation of recursively defined subroutines capable of operating on data of arbitrarily complex structure. This paper will develop, partly from a historical point of view and partly for the benefit of a programming novice, the requirements placed on the data implementation, and the usefulness of the data structure to symbolic computation. A self-contained and motivating data presentation for the novice has not been adequately handled elsewhere, as previous works invariably define a classic logical language of well-formed-formulae over a character alphabet — an approach which does not relate well to the structured nature of LISP data, and which cannot provide the basis for explaining one of the primary data predicates: EQ. In addition, the goal of embedding the programming language into the data language, and achieving efficient interpretation therein, will be discussed. LISP is unique in that a simple data operation will take an expression of the data language and, leaving its structure intact, extend it to be an applicable function in the programming language. This is essentially the ability to create LAMBDA expressions dynamically (and, where appropriate, to create FUNARG expressions, and to compile functions at run time). It is not expected that this paper will be sufficient for a novice actually to learn how to program in LISP, but it should provide a good, basic understanding of the concepts involved.

## THE DATA

### Its Structure

In many programming languages, the data are essentially "flat" objects. In FORTRAN, the basic datum is an integer (or floating point number), limited in information content to some fixed number of bits, and the basic arithmetic operators are not thought of as decomposing an integer into sub-parts. Even the notion of a vector of numbers is quite "flat" since the components of such a vector are not themselves considered to be sub-vectors, but merely numbers. In languages which provide for character-string processing, there is a similar "flatness", with 'number' replaced by 'character', and 'vector' replaced by 'string'. Just as we would not want each program variable to be restricted to one kind of data, similarly we would not want our most general type of composite data to be restricted as to the type of subcomponents it may have. Another problem in these languages is that the program variables must often be restricted to data of a particular size --- FORTRAN integer variables being implicitly limited by the word size of the supporting machine, FORTRAN vectors (and vector variables) requiring explicit compile-time dimensioning of sizes, and PL/I string variables being limited analogously by explicit program declaration.

---

*During the calendar year 1977, the author is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, and wishes to acknowledge members of the LISP370 project as having contributed to the development of ideas in this paper.

One goal of LISP is to remove the limitations of "flatness" and size from the data objects and their corresponding variables; e.g., typeless variables are permissible in LISP, and the transition from hardware-supported integer arithmetic (modulo, say, $2^{35}$) to infinite-precision integer arithmetic need not concern the programmer (except for the question of computation cost). For the data to be of the most general structure, its components must not be restricted as to type; in short, the data should be defined recursively. Two obvious features of structured data sets are: 1) that at least some of the data structures have more than one component (otherwise, there would be no structure!), and 2) that without any real loss of generality it is sufficient to have only binary structures, since there is a natural, easy embedding of any other into these.

LISP has, for its basic non-atomic data, objects of two components which are decomposed by the functions CAR and CDR, and which are built up by the function CONS. These functions represent, in an abstract sense, the necessary operators defined over a structured data set — CONS being mnemonic for the construction function, and the other two, subcomponent accessors, being named after a particular feature of the architecture of the IBM 704 on which the first LISP system was implemented. In fact, actual machine architecture has deeply influenced LISP design, for one goal of LISP was to become a useful programming language. Thus, a first step was to assign a logical record of memory (that is, some finite number of bits easily accessible by the supporting hardware) to hold a data object; we call such a block of memory a "cell", and use the machine address of the cell as a handle for the object. An address used this way will variously be called a "pointer" or "name" of the stored object. Half of the bits in the cell (or thereabouts) hold the first part of the pair, accessed by CAR, and the other half hold the second, or CDR, part. Computer architecture intrudes at this point, in that the computer word is often chosen as the unit of memory for a cell, partly because of economy in memory utilization and partly because of a computer instruction repertoire which permits easy decomposition of data stored this way. This has been true for almost all PDP10 LISPs, and quite a few IBM360 LISPs, but LISP370 (an experimental LISP at IBM's Research Center) uses a double word for each cell, and the MULTICS MACLISP takes four words per cell. At first, this storage method seems to invalidate the goal of not limiting the size of a data object to a fixed bound, but this is not nearly so serious as it may seem, since the parts of a cell are interpreted as names for other cells; thus a data object is thought of as a graph, consisting of all the cells and links reachable from a given pointer by CAR and CDR.

In the world of algebraic manipulation, any reasonable fixed allocation for the maximum size of integers will prevent most simplification algorithms from working.[1] For this reason, most good LISP systems provide for variable-precision integer arithmetic, often by embedding the parts of a long integer into one of the other complex data structures. However, the maximum size of a data structure is limited by the total number of names available for nodes of the conceptual graph which it represents, and this name space is limited by the number of bits in a half-cell. At the outset of LISP development, large computers had up to 32K words of main memory, and this was thought to be larger than any program would ever need; however, applications soon came up requiring many times that number of LISP cells

---

[1] An "unreasonable" size allocation would be one in which only a few hundred integers could fit in main memory at one time. The default allocation for most languages is one computer word per integer, because there is generally built into the hardware the circuitry for quickly doing arithmetic on one- or two-word cells. One can only go so far in attempts to speed up arithmetic with larger and larger circuitry, as the work of Winograd shows in references 1 and 2. Another approach at increasing speed has been to analyze numerical algorithms, trying to separate out the parallel parts so that duplicate arithmetic units may carry out the subcomputations in parallel; the ILLIAC-IV has much circuitry involved in the latter approach.

— MACSYMA is a particularly good offender in this regard. An early LISP at the IBM Research Center had only a 16-bit address space, and was soon "choked" to death by SCRATCHPAD the current system, LISP370, has a 24-bit address space in a completely revised design. This size seems optimistic now (24 bits, of which three are the byte address within a cell, leaving room for addressing 2M cells), in that 2 million 64-bit doublewords is probably more main memory than most computers are likely to have directly addressable during the next five or so years, but we have been wrong about this in the past. The danger of biting off too many bits for the address space is that each cell would then require more and more words for storage, and thus with a bounded amount of main memory fewer and fewer cells could be held therein. Of course, $2^{<\text{number of address bits}>}$ is a real upper bound, even in a virtual-memory machine with a much smaller amount of real memory. Sometimes, it is possible to segment the data and process it in two or more passes so that it need not all be directly addressable at once, but the familiar "intermediate-expression swell" of algebraic manipulation shows that this can not serve as a general solution. Again, it would be possible to extend the name space beyond actual address space by treating each name as an address in an extended secondary-storage space; however, except for very limited applications, this would slow down operations drastically. The costs of computer memories are still decreasing, larger and larger address spaces are becoming more feasible, but the finiteness bound is still there. Even though we have bumped into the top of that bound several times, it should not be too frightening: an excellent article by Knuth puts "finite" into proper perspective (reference 3).

A data object, graphically represented as in figure 1, can easily and directly be translated into computer memory by assigning each node of the graph a new cell, and labelling each directed edge with the address of the translated node that it points to. Stored in a cell, then, would be the two addresses found on the edges leading out of the corresponding node. In order to get these data "off the ground", certain structures are designated as atomic, that is, not decomposable by (there are no sub-parts accessible by) the functions CAR and CDR. Atomic objects can be denoted graphically as a string of alphabetic characters (from a computer alphabet such as ASCII or EBCDIC), and in figure 1 they are enclosed in rectangular rather than round boxes.[2] The collections of atomic and non-atomic data are called "s-expressions", which is short for "symbolic expressions".

### Atoms — Symbols

Atoms are in fact structured objects (but not in the general sense described above), and their sub-parts are obtained by specialized accessor functions. Because of the varying potential for efficiency of representation and operation, there are generally several classes of atoms in a LISP system, distinguishable in their memory structure. A most important one of these will be called an "atomic symbol", or merely SYMBOL, and each has a place in its structure for storing (i) a pointer to a list of associated properties, (ii) a pointer to a binding cell when the symbol is being used as a program variable, (iii) a string of alphabetic characters for denoting the object on input-output, and possibly other parts depending on the implementation. Item (iii) has been historically called the print name, but now generally acronymized as PNAME (pronounced pea-name), and provides the output routine with a quick method of generating a sequence of characters corresponding to that object. An input routine, when given a string of characters, could, by taking new cells of storage, construct a symbol with that string as PNAME. But more often, it is desired to use the PNAME sequence as an external, address-free reference to a specific symbol, a canonical symbol with that PNAME, so that pre-existing properties attached to that particular object may be easily accessed — the PNAME thus serving as a kind of "key". The standard input routine for LISP, generally called READ, constructs s-expressions by parsing an input stream of characters; but in particular, when it parses a string into a PNAME, it uses a function INTERN to locate the canonical symbol with that PNAME; INTERN, in turn, accomplishes this by keeping a table (called the OBARRAY, or the OBLIST) of all canonical symbols, creating new ones as the need arises. Some implementations do not permit the creation of any symbols except the canonical ones, so that no two distinct symbols would have the same PNAME; but in others not so strict, the terminology "uninterned atom" is used to mean a symbol not entered (and hence not "canonical") on the current OBARRAY. The importance of an external, address-free reference will be seen as this paper develops the presentation of the LISP data language as a programming language: atomic symbols are used as names (in the informal sense) for system subroutines, for user-defined subroutines, for program variables, and for a few specially recognized constants.

---

[2] Our use of rectangular and round boxes is an inversion of the convention found in other presentations, e.g. Weissman's "LISP 1.5 Primer" (ref. 4), and the "LISP 1.5 Programmers Manual" (ref. 5). This is by design, partly to emphasize that the structure in the boxes, rather than their shape, is the important thing, but also two other advantages occur: 1) the PNAMEs of atoms, which can be quite long, have a box shape more suitable to their typography, and 2) there is a fuller separation between the older notation, which prompted one to think of s-expressions as well-formed-formulae over a character set, and the notation in this paper, which only begrudgingly admits of the linearized print form.

### Atoms — Numbers

The desire to use machine hardware arithmetic instructions, and to economize on storage, has led LISP to introduce the class of atoms called FIXNUM (and, in most systems, FLONUM also). The programming language provides basic predicates for testing whether a given object is an atom of numeric type, the most general such being NUMBERP, and most LISP systems support a variety of numeric data types with associated type-specific predicates in order to accommodate programming needs (some LISPs also provide a basic predicate to test whether an object is an atomic symbol, such as SYMBOLP in MACLISP and LITATOM in INTERLISP, but some others do not — the programmer resorting to a compound form like "atom[x]∧~numberp[x]"). A fixnum, for example, has a word in which a number is stored in the usual computer notation (say, 2's complement in a 36-bit word); numeric operations will now be facilitated, but the output routine will have to go through some base-conversion process to produce the digit-string that one would like to see for that number. On the input side of the question, a digit-string can be evaluated assuming a particular radix notation, and a new cell (or cells, if a multiple-precision integer is indicated) allocated for storing the incoming number. At this point, a certain ambiguity is evident concerning the input parser: should a string of characters, all of which are decimal digits, be converted into a fixnum, or into a symbol with that string as PNAME? As a convention, such a string would be input as a fixnum (or flonum if the sequence also had some character recognized by the parser as a floating point indicator), and another convention is established for escaping the special significance that the parser might apply to particular characters. In MACLISP, the character / is used in prefix of any character that might otherwise cause the parser not to include that character in the PNAME of a symbol. For example,

1729

could be read in as a fixnum, the least integer expressible as the sum of two cubes in precisely two different ways, whereas

/1729

would be read in as a symbol with four characters in its PNAME. There are no systemic properties associated with a number other than its numerical value, so there seems to be no need to try to identify a canonical storage location for a given value (but some systems do canonicalization, of varying degrees, in order to reduce storage utilization).

### Lists

The general data structures of LISP are then built up over the field of atomic objects with the construction function CONS. The basic non-atomic object, because of the way it is constructed and stored, is called by some persons a "cons" cell, by others a "pair", and by many others a "list" cell. As a function, CONS is anti-commutative in that if $e_1$ and $e_2$ are unequal, then CONS[$e_1$,$e_2$] and CONS[$e_2$,$e_1$] are also unequal. Graphically, this is seen in figure 1 in that the edges emanating from a node have a definite left-hand and right-hand orientation; also evident is the binary nature of CONS, in that each non-atomic node has precisely two edges emanating from it (and each atomic node has none).

The external, linearized representation of a non-atomic object; called its "print representation", is a modification of a fully-parenthesized notation. The full notation is easily described: let $e_1$ and $e_2$ be any two data objects, and let $e_1^{\bullet}$ and $e_2^{\bullet}$ be their respective print representations. Then a data object constructed from $e_1$ and $e_2$, that is by CONS[$e_1$,$e_2$], will have the print representation

$$(e_1^{\bullet} \ . \ e_2^{\bullet})$$

It is generally convenient to think of the pair cell as holding a list, even though this is only an interpretation in the mind of the beholder: the CAR part of a pair is the first element of the list, and the CDR part is the tail of the list with the first element removed. Ostensibly, by successive applications of the CDR function, some atom will be reached; by convention, we desire this atom to be the symbol NIL, and elevate it to the status of the null list, i.e., the list with no elements. Many LISP systems will permit list operators to work with lists terminating in some other atom, but by fixing on this conventional use of NIL, the following simplification can be made for the print representation:

(i) Instead of $(e_1^{\bullet}$ . NIL),
we will print $(e_1^{\bullet})$

(ii) Suppose there is a list *l* which prints as
$(e_1^{\bullet}\ e_2^{\bullet} \ldots en^{\bullet})$,
then, for $l'$ = cons[$e_0$,*l*], instead of
$(e_0^{\bullet} \ . \ (e_1^{\bullet}\ e_2^{\bullet} \ldots en^{\bullet}))$,
we will print $(e_0^{\bullet}\ e_1^{\bullet}\ e_2^{\bullet} \ldots en^{\bullet})$

Figure 2 shows a graph for a data structure, as in figure 1, with the two possible print representations printed below it. Note, also, the several common references to the boxes for the symbols ABC and NIL, and the duplication of the boxes for the fixnum 35; see how the graph more directly shows the canonicalization that has taken place for the input of symbols and the duplication for input of numbers.

## THE PROGRAMS

What kind of operations might one want to do in this data world? McCarthy's classic paper, "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I" (ref. 6, one might say the grandfather of LISP papers), is a good start at answering this question. Both it and the LISP 1.5 Programmer's Manual indicate that the elementary operations CAR, CDR, CONS (discussed above as being the requisite operations needed over any binary structured data set), and the elementary predicates ATOM, EQ, along with the mathematical notions of functional composition, conditional expression, and recursive definition comprise a sufficient means to build up any computable function on this data domain.[3] This collection of primitive functions and functional schemata is minimal in that no one part can be derived from the others alone. (The two points, sufficiency and minimality, have been proven by Mike Levin, one of the early originators of LISP). Of course, in real usage, many more functions are added for the convenience of the programmer; part of the job of a LISP system implementer is to choose a reasonable set of basic, system-supplied functions — not so large as to bloat the computer's memory, and not so small as to unduly cramp the programmer.

Historically, the development of LISP as we know it today, was quite accidental. Originally, it was assumed that various functions could be defined and written down with some mathematical rigor, using a more-or-less standard mathematical notation which was called the Meta-language (see refs. 5,6). Then from this presentation, one would compile the algorithm into a machine language program, with subroutines holding their data and exit addresses on a stack in order to provide for recursive operation

---

[3] It is interesting to note that the paper (ref. 6), while laying the foundation of a good non-numeric *data structure* for computers (symbolic expressions), at the same time has had a profound effect on the development of *program schema*, namely the way in which programs are put together from components. Conditional expression and memory operation are required in any non-trivial programming world, but McCarthy, by emphasizing functional composition and recursive definition, injected a bit of mathematical common-sense into the world of sequential programming.

— hardly the interpretive LISP we know today! In any programming project, the task of getting programs into the computer always becomes more difficult as time goes on (and time has a notorious infamy for always going on), so someone had the bright idea of transcribing programs, not into machine language, but into the data language already defined, namely s-expressions, so that they could be automatically translated into machine language. The first mechanical compiler was, of course, written in machine language, but it was not very successful (needless to say, subsequent compilers were written in LISP). Then, one of the programmers associated with the original LISP project had the bright idea of making an s-expression evaluator, which could interpret these encoded programs, and hence, through EVAL, the LISP interpreter was born.

That single idea has had enormous consequence on the development of the fields of list processing, artificial intelligence, and symbolic manipulation. Although some other languages, such as APL, permit the dynamic evaluation of computed expressions, in none save LISP is the programming language so thoroughly embedded into the data. In no other is there the smooth naturalness with which LISP programs may dissect, analyze, report upon, review, "dress-up", synthesize, emulate, and compile other LISP programs.

## Functions, Functional Composition, and QUOTE

What, then, is the transcription scheme? It is really quite simple. First, we note that most LISP systems have at least the characters of the 6-bit ASCII alphabet, which is 26 uppercase letters, 10 digits, some punctuation marks, and the usual assortment of special characters found on most typewriters or teletype machines. Then, a variable or function is represented by the symbol of the corresponding PNAME; numbers stand for themselves, that is they will be transcribed directly; functional application is shown as a list of the function and all its arguments in order; functional composition is shown as list composition; the elementary operations are represented by the atomic symbols CAR, CDR, CONS, ATOM, and EQ; and some of the basic arithmetic operators are implemented with mnemonic names in prefix notation (instead of writing "x+z+2.3", we would write in prefix notation "plus[x,z,2.3]"). As an example illustrating all the rules mentioned so far, we would transcribe

$$5\bullet[\log \sin(x+z+2.3)]$$

into a list printable as

(TIMES 5 (LOG (SIN (PLUS X Z 2.3)))) (1)

If all our functions were defined only over numbers, then the intent of such a program, coded in list structure, is clear: add together the numeric values of the variables x and z and the number 2.3, take the trigonometric sin of the result, then the natural log of that, and finally multiply by 5. But some of our functions are defined over lists as well as other objects, and the question arises as to how the argument for such a function is obtained. For example, suppose we want to print out the list (PLUS X 3), and suppose coincidentally that the variable X has the value 7. Then what does

(PRINT (PLUS X 3)) (2)

do as a program? By the above rules, it should print out the number 10. How then are we to indicate that we want to print out the list (PLUS X 3)? It becomes necessary to add a rule in the transcription scheme that overrides the notation for functional composition — for this purpose, we use the atom QUOTE in the first element of a list to indicate that the second element is not a sub-program, but rather is to be taken directly as data without any interpretation. Line (2) above would print out the number 7, whereas

(PRINT (QUOTE (PLUS X 3))) (3)

would print out the desired list, (PLUS X 3). Line (2) could be a transcription of the expression "print[x+3]", whereas line (3) could be that for "print['(PLUS X 3)']".

There are several kinds of overrides to the functional composition rule, to be discussed in turn below. Because of the similarity of structure — namely, an atomic symbol at the first element of a list — many persons have begun referring to these overriders as "functions" also; but they should more properly be viewed as parts of the syntax of the programming language LISP. In LISP 1.5, they are called "special forms". In particular, they represent the realization in LISP of some of the abstract, universal concepts found in any practical programming language; e.g., COND, PROG, SETQ, DEFINE. LISP further has QUOTE as just discussed, and LAMBDA — the former to distinguish data expressions from programs in which the data might be embedded, and the latter to distinguish programs from some data in which they, in turn, might be embedded. At this point, it must be stressed that these rules and conventions comprise part of the *programmatic interpretation* of LISP data expressions; other, radically different interpretations are possible, e.g. without QUOTE, or without PROG and SETQ, but they are generally less usable.

Program interpretation also implies an importance to the sequence in which the sub-computations are carried out. If there were no *memory* cells in a computer, nor any *side-effects* during computation, then the order of evaluation of the sub-parts of a program would be irrelevant. For example, what difference would it make if, in computing "(x+3)•(y-5)", the sum were performed after the difference calculation? Logically, none, but if while computing the difference "y-5", some action is taken that changes the value of the variable x, then probably a different final product would result. The normal rule for LISP program interpretation is left-to-right order of evaluation, beginning with the first element of the list. This first element, corresponding to some function to be applied, is inspected for a basic function definition, or for one supplied by the programmer (which may involve recursion through the interpreter),[4] and then the first argument to the function is calculated according to the program part in the second element of the list; and then the third, and so on. Finally, the function is invoked with the corresponding arguments. The special forms PROG and SETQ do not come under this normal rule. PROG corresponds to the sequential nature, with GOTOs, of FORTRAN programs; and SETQ corresponds to the notion of assigning a new value to a variable while releasing the old value. Because of lack of space, these features will not be further discussed in this paper.

### Predicates and Conditional Expressions

Predicates operate on data to produce one of two values — *true* or *false*. In the LISP world, we let the symbol NIL encode the value *false* and T encode *true*. However, as a convenience, we allow any non-NIL value to be returned by a predicate, and in so doing interpret it as *true*. Furthermore, we remove NIL and T from the collection of possible program variables, considering them as constants which stand for themselves just as numbers do.

The elementary predicate ATOM is a function which is true for terminal nodes of the graph-structured data (the items in rectangular boxes in figures 1 thru 3), and false for cons cells. It is apparent that the domain of ATOM on which it is *false* is precisely the domain of s-expressions on

---

[4] Normally, the identity of the function, or sub-program, to be applied is evident upon "inspection", in that it will be an atomic symbol with some direct functional property. What happens when this is not the case has never been clearly defined — notice, for example, the discrepancy between lines 18-19 and line 20 on page 71 of the LISP 1.5 Programmer's Manual (ref. 5); and reference 6 has an even more confusing bug at the corresponding spot of the definition of EVAL. Most LISP systems make one evaluation of the first element, then evaluate all the remaining elements once in order to obtain the arguments, and then begin a process of re-evaluation of the result from the first element until it is directly discernible to be a function. There is no problem unless some relevant memory location is changed, such as happens in the following example. First, note the shorthand convention of writing '*exp* instead of (QUOTE *exp*)

```
((SUBST 3 'N '(PROG2 (SETQ X (PLUS X N)) 'DIFFERENCE))
 X
 Y)
```

In this case, by evaluating the first element successively twice, one gets a result different from that obtained by the order of evaluation just mentioned above.

which CAR and CDR are applicable. Atoms which are interpretable as numbers are stored in computer memory in such a way as to require specialized functions and predicates, for the purpose of achieving efficiency in numeric operations; e.g., NUMBERP, FIXP, FLOATP, GREATERP, and numeric-equal. In MACLISP, and some others, many new numeric functions and predicates have been introduced generally having shorter names, such as > as a less general form of GREATERP, = for (exact) numeric equal, + for addition restricted to fixnums, +$ for addition restricted to flonums, and so on.[5]

The predicate EQ, a function of two arguments, is a test for pointer identity, let us see how this works. In figure 3, two lists L1 and L2 are shown graphically along with their print representation (in L2, the edges are not shown as extending all the way to the rectangular boxes for atoms, merely because of the complexity of drawing too many intersecting lines). Suppose for example that the top node of L1 is stored in a cons cell at computer address 0129, and L2 at 3724. Let x, y, z be program variables such that x =L1, y = L2, and z = L1. This means that the variables hold some pointer to a cons cell — the bits of x and z would correspond to the decimal number 129, and those of y to 3724. But a LISP system *interprets* this pointer according to its data classification; thus ATOM is *false* for each of the variables, and each would be printed out as

```
(LIST (QUOTE FOO))
```

Now, EQ is *true* of [x,z], but *false* of [x,y] and [y,z] because x and z hold the same pointer, but x and y are different pointers corresponding to isomorphic structures.

Of course, not all functions, even over the domain of numbers, are smooth and "analytic"; discontinuities of various sorts can be introduced by conditional expressions. Let DELTA be defined as a function of x and n as follows: 1 if x>n, -1 if x<n, and 0 otherwise. This conditional expression would be transcribed into LISP as

```
(COND ((GREATERP X N) 1)                    (4)
      ((LESSP X N) -1)
      (T 0))
```

As with QUOTE, COND is a special form in the programming language, and indicates that a sequence of sub-lists follows, each sub-list consisting of one or more expressions. The first elements of the sub-lists are evaluated in sequence order until the first one that comes up not *false* is found; the remaining elements of that sub-list are then evaluated and value of the last element (which might also incidentally be the first) is taken as the resulting value for the COND expression. In addition to the "discontinuity" which the conditional expression introduces, there is a noticeable programmatic feature, namely that of *selective* evaluation. Not all of the predicates are evaluated, but only those which, in sequence, turn out to be *false*, up until the first one that is *true*. Obviously, COND may be thought of as a compound predicate; so are OR and AND, whose definitions are in accord with one's intuitive notion. It may be helpful to see corresponding code for OR and AND in terms of COND:

```
(OR x y z)          (COND (x)
                          (y)
                          (T z))

(AND x y z)         (COND (x (COND (y z)
                                   (T NIL)))
                          (T NIL))
```

To round out the logical connectives, NOT operates as truth-value inversion. Both (NOT *x*) and (NULL *x*) operate the same as the expression (COND (*x* NIL) (T)).

---

[5] LISP systems [illegible] have introduced novel data types generally have introduced functions and predicates with [illegible]stricted domains in order to operate efficiently on them. This is one way of extending LISP.

## Defining Functions

The expression (4) above is almost a definition for a function "delta", but it is not symmetric in the two variables x and n; if you were to write (DELTA 3 5), you would want to know whether X would hold 3 and N 5, or vice-versa. The symbol **LAMBDA** is a special form to indicate that a function is being defined from an expression, by specifying the order in which the variables of the expression shall correspond to the incoming arguments. Rewriting (4) as a functional expression, we get

```
        (LAMBDA (X N)                                                   (5)
                (COND ((GREATERP X N) 1)
                      ((LESSP X N) -1)
                      (T 0)))
```

Now (5) is an expression that can be applied to [3,5] and result in -1, but when applied to [7,2] results in 1. The syntax permits us to write this expression directly in the functional position of a list intended for program interpretation:

```
        ((LAMBDA (X N) (COND ((> X N) 1) ((< X N) -1) (T 0)))  3  5)
```

However, for convenience of writing, we might like to define **DELTA** as a function name corresponding to the functional expression (5); in the case of recursive definition, there is no choice about the matter, we must start out with some function name so that we can write down the definition using that name. Consider the classic case, defining the factorial function.

```
        (LAMBDA (N) (COND ((= N 0) 1) (T (* N (fact-continuation (- N 1))))))
```

At the point where *fact-continuation* occurs, we would like another copy of the entire functional expression substituted, so that the computation could be carried on recursively. Rather than extend the notation to encompass cyclic structure, or to infinite sub-structure, we find that using a symbol as a name for a function being defined solves not only this problem, but also that of conciseness. Thus the factorial definition becomes:

```
    (DEFINE FACT (LAMBDA (N)                                                    (6)
                         (COND ((= N 0) 1)
                               (T (* N (FACT (- N 1)))))))
```

Function definition is generally realized in a **LISP** system by executing a program that places a property on the property list of the symbol which is the function name; **DEFINE** (or **DEFUN** in **MACLISP**) is a special form which causes this to happen.[6] Evaluating (DEFINE FOO *exp*) will cause an attribute-value pair to go on the property list of FOO — the attribute name is **EXPR**, and its corresponding value is *exp*. The interpreter can then quickly recognize FOO to be a function name by accessing its **EXPR** property, and substituting the **LAMBDA** expression so obtained for the name. In the case of machine-language subroutines, a starting-address is stored under the **SUBR** attribute, and, after the arguments are obtained, the interpreter can quickly despatch control off to the relevant location. In such a **LISP**, one needs only the ability to read-in lists and to evaluate them after read-in in order to add subroutines (or programs, if you will) to the system. The so-called "top level" of a **LISP** system is basically a loop:

```
            A: print(eval(read()))
               go A
```

From this we can see the importance of **INTERN** to the input **READ** function: it is necessary that both instances of "FACT" in (6) above be read in as pointing to the same atomic object (and not merely to atoms with the same **PNAME**), and the same holds true of the three instances of "N". Thus it is that one programs in **LISP**, and interacts with **LISP** environment.

---

[6] There are **LISP** systems that do not use the property list for function definition, but instead use whatever mechanism implements the assignment of a value to a variable. This approach is adequate, although it means that one could not use a symbol both for a variable name and a function name.

### A Useful Example

Let us consider a definition of an "equality" predicate EQUAL defined over all the data types mentioned in this paper, such that two s-expressions are printed out in linear format the same way if and only if they are EQUAL. For numbers, the numeric equality predicate is used; for symbols, SAMEPNAMEP and for lists, the definition is recursive over the CAR part and the CDR part. Historically, EQUAL was defined before any consideration was given to multiple copies of atomic objects all with the same PNAME; hence EQ was generally used instead of SAMEPNAMEP; because if two symbols were stored in different locations then they necessarily had different PNAMEs. As far as the author knows, all LISP systems still use EQ here, and this is considered satisfactory.

```
(DEFINE EQUAL
 (LAMBDA (X Y)
   (COND  ((EQ X Y) T)
          ((ATOM X)
           (COND  ((NOT (ATOM Y)) NIL)
                  ((AND (NUMBERP X) (NUMBERP Y)) (= X Y))
                  ((OR (NUMBERP X) (NUMBERP Y)) NIL)
                  (T (SAMEPNAMEP X Y))))
          ((ATOM Y) NIL)
          ((EQUAL (CAR X) (CAR Y)) (EQUAL (CDR X) (CDR Y)))
          (T NIL))))
```

It would be instructive for the reader to consider this example line by line to verify how it works. Note carefully that EQUAL does not define "graph-isomorphism", but rather a concept that has come to be called "access-equivalence". Two structures are said to be access-equivalent (or EQUAL) if any access chain (a sequence of CARs and CDRs, for LISP) leading to an atomic object in one structure also leads to the same atomic object in the other. See figure 4 for a graphic presentation of two structures that are EQUAL but not isomorphic.

## REFERENCES

1 Winograd, S., On The Time Required to Perform Addition. *J. ACM 12, 2*, April 1965, pp. 277-285.
2 Winograd, S., On The Time Required to Perform Multiplication. *J. ACM 14, 4*, Oct 1967, pp. 793-802.
3 Knuth, D., Mathematics and Computer Science: Coping with Finiteness. *Science, 194*, 17 Dec 1976, pp. 1235-1242.
4 Weissman, Clark; *LISP 1.5 Primer*, Dickenson Publishing Co., 1967.
5 McCarthy, John, et. al.; *LISP 1.5 Programmer's Manual*, The MIT Press, Second edition 1965.
6 McCarthy, John, Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I. *CACM 3, 4*, April 1960, pp. 184-195.

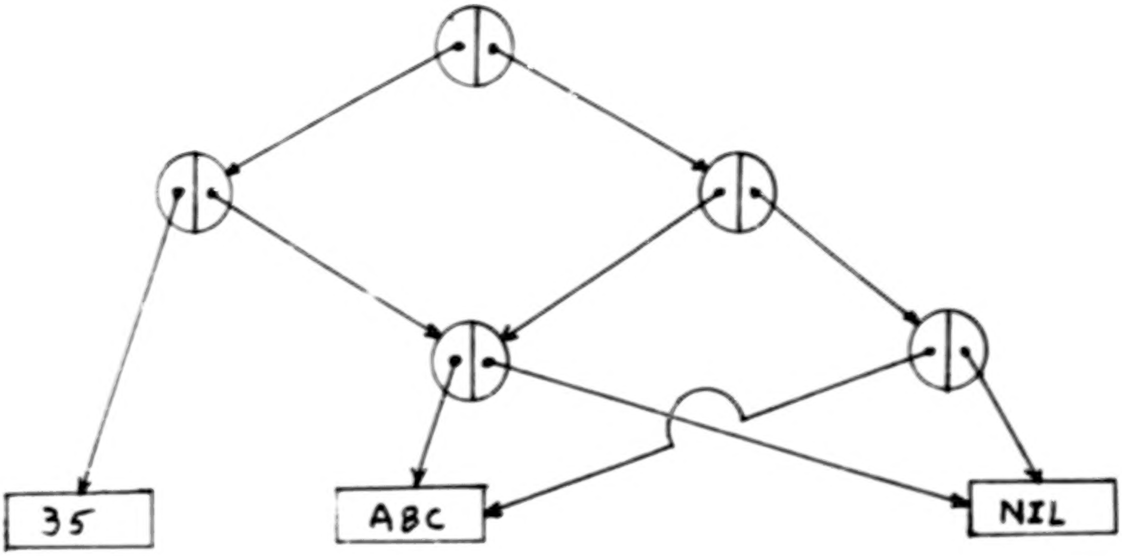
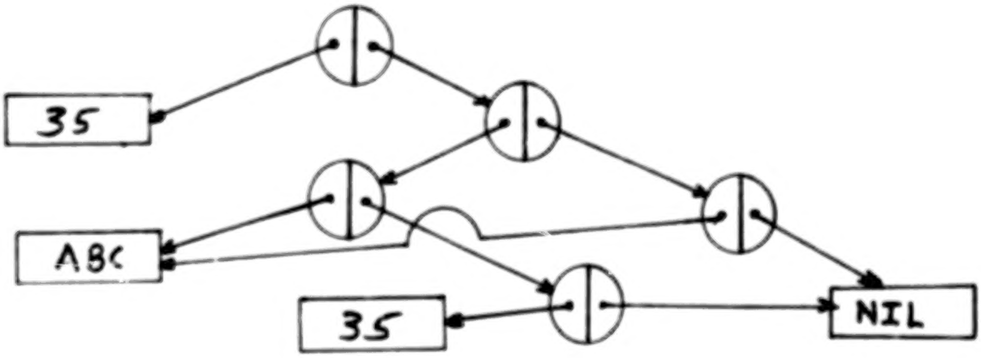


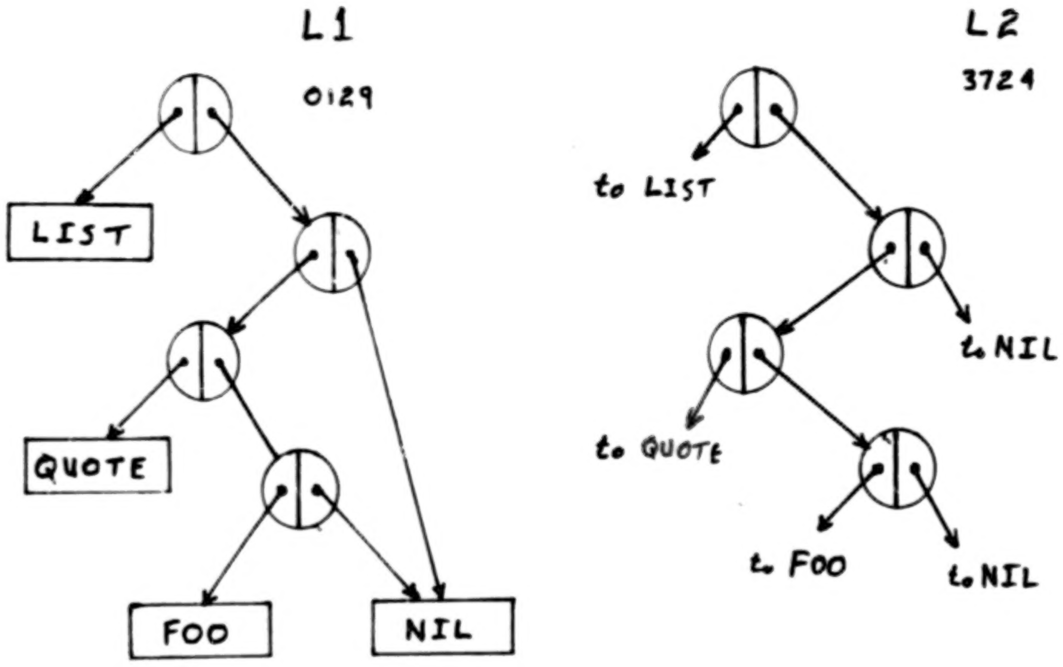
Figure 1


35
ABC
35
NIL

(35 . ((ABC . (35 . NIL)) . (ABC . NIL))

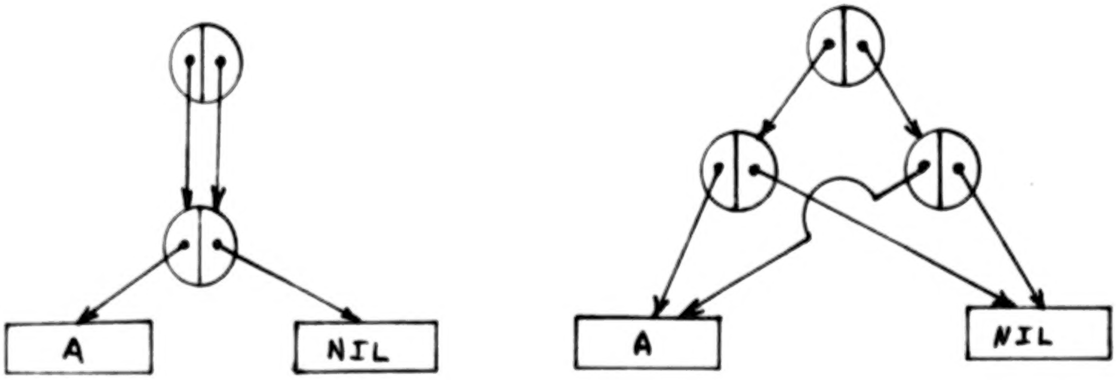
(35 (ABC 35) ABC)

Figure 2


L1
0129
LIST
QUOTE
FOO
NIL
L2
3724
to LIST
to NIL
to QUOTE
to FOO
to NIL

(LIST (QUOTE FOO))     (LIST (QUOTE FOO))

Figure 3



((A) A)     ((A) A)

Figure 4

# DATA REPRESENTATIONS IN PDP-10 MACLISP

Guy Lewis Steele Jr.
Massachusetts Institute of Technology
Laboratory for Computer Science
(formerly Project MAC)

## ABSTRACT

The internal representations of the various MacLISP data types are presented and discussed. Certain implementation tradeoffs are considered. The ultimate decisions on these tradeoffs are discussed in the light of MacLISP's prime objective of being an efficient high-level language for the implementation of large systems such as MACSYMA. The basic strategy of garbage collection is outlined, with reference to the specific representations involved. Certain "clever tricks" are explained and justified. The "address space crunch" is explained and some alternative solutions explored.

## INTRODUCTION

MacLISP is a version of LISP which is used not only as a user application language but as a systems programming language, supporting such systems as MACSYMA and CONNIVER. As such, it has been carefully designed with speed as one of its major goals. Generality, ease of use, and debuggability have not been neglected, but speed of compiled code has been the primary consideration. This is a departure from the traditional view of LISP as a friendly and general but slow and clumsy language.

The representations of data objects in MacLISP have undergone a continuous evolution towards this goal. When MacLISP was first created, the data representations were designed for simplicity and compactness at the expense of speed. Since then there have been at least two major revisions, each to speed up compiled code and simplify the processing of the data. Here we discuss the current implementation on the PDP-10 (MacLISP also runs on Multics, and on the "LISP machines" being constructed at the MIT Artificial Intelligence Laboratory). We shall contrast it with previous MacLISP implementations and implementations of other LISP systems, and discuss some of the design decisions involved.

## ORGANIZATION OF THE PDP-10

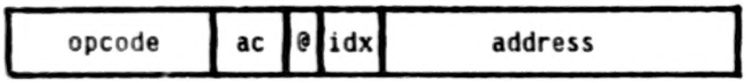
The data representations in MacLISP have been carefully designed to take full advantage of the PDP-10 architecture. A full understanding of the design decisions involved requires the following minimal knowledge of the PDP-10 instruction set.

The PDP-10 operates on 36-bit words. Memory addresses designate words, not bytes, and are 18 bits wide; thus two addresses can fit in one word. There is a class of instructions which manipulate half-words; for example, one can store into half of a memory word and either not affect the other half or set the other half to all zeros or all ones.

The PDP-10 has 16 accumulators, each 36 bits wide. All but one can be used for indexing; all can be used as stack pointers; all can be used for arithmetic.

The accumulators can also be referenced as the first 16 memory locations (though they are hardware registers and not actually memory locations). For reasons explained later, MacLISP devotes certain accumulators to specific purposes. Accumulator 0 contains the atom NIL. Accumulators 1-5 may contain pointers to data objects; these are used to pass arguments to LISP functions and return values from them. Accumulators 6-10 are scratch registers, and are generally used for arithmetic. Accumulator 11 is reserved for a future purpose. Accumulators 12-15 are used for stack pointers to the four stacks.

Every user PDP-10 instruction has the following format:

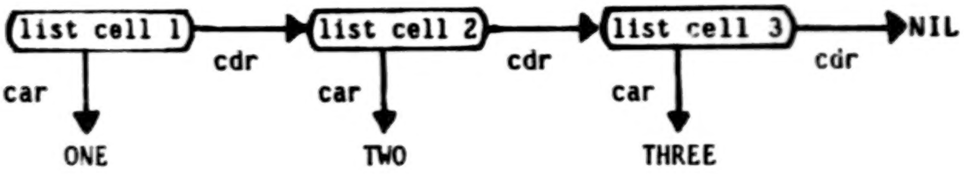


Each instruction has a 9-bit operation code and a 4-bit field specifying an accumulator. The effective memory address (or immediate operand) is uniformly computed by adding to the 18-bit address field the contents of the accumulator specified by the 4-bit index field (a zero index field means no indexing). If the indirection bit "@" is set, then a word is fetched using the computed address and the process iterated on the address, index, and @ fields of the fetched word. In this way the PDP-10 allows multiple levels of indirection with indexing at each step.

## MACLISP DATA TYPES

MacLISP currently provides the user with the following types of data objects:

| | |
|---|---|
| FIXNUM | Single-precision integers. |
| FLONUM | Single-precision floating-point numbers. |
| BIGNUM | Integers of arbitrary precision. The size of an integer arithmetic result is limited only by the amount of storage available. |
| SYMBOL | Atomic symbols, which are used in LISP as identifiers but which are also manipulable data objects. Symbols have value cells, which can contain LISP objects, and property lists, which are lists used to store information which can be accessed quickly given the atom. Symbols are written as strings of letters, digits, and other non-special characters. The special symbol NIL is used to terminate lists and to denote the logical value FALSE. |
| LIST | The traditional CONS cell, which has a CAR and a CDR which are each LISP objects. A chain of such cells strung together by their CDR fields is called a list; the CAR fields contain the elements of the list. The special symbol NIL is in the CDR of the last cell. A chain of list cells is written by writing the CAR elements, enclosed in parentheses. A non-NIL non-list CDR field is written preceded by a dot. An example of a list is (ONE TWO THREE), which has three elements which are all symbols. It is made up of three list cells thus: |



| | |
|---|---|
| ARRAY | Arrays of one to five dimensions, dynamically allocatable. |
| HUNK | Short vectors, similar to LIST cells except that they have more than two components. This data type is fairly new and is still experimental. |

## POINTERS

In MacLISP, as in most LISP systems, the unit of data is the pointer. A pointer is typically represented as a memory address, with the components of the data object pointed to in the memory at that address. The reason for this is that LISP data objects have varying sizes, and it is desirable to manipulate them in a uniform manner. Numbers, for example, may occupy varying numbers of words, and it is not always feasible to put one as such into the accumulators. A pointer, being only 18 bits, can always fit in one accumulator regardless of the size of the object pointed to; moreover, it requires only 18 bits for one data object to contain another, since it need actually only contain a pointer to the other.

Given a pointer, it is necessary to be able to determine what kind of object is being pointed to. There are two alternatives: one can either have a field in every data object specifying what type of object it is, or encode the type information in the pointer to the object. The latter method entails an additional choice: one can either adjoin type information to the memory address (in which case it takes more bits to represent a pointer), or arrange it so that the type is implied by the memory address itself (in which case the memory must be partitioned into different areas reserved for the various data types). MacLISP has generally used this last solution, primarily because of the half-word manipulation facilities of the PDP-10. Two memory addresses will fit in one word with no extra bits left over. (Contrast this with an IBM 370, which has 32-bit words and 24-bit addresses; on this machine one would use 32-bit pointers, encoding type information in the extra eight bits.) This is extremely useful because a list cell will fit in one word; the left half can contain a pointer to the CAR and the right half a pointer to the CDR.

The method MacLISP presently uses for determining the type of a data object involves using a data type table. The 18-bit address space (256K words) of the PDP-10 is divided into segments of 512 words. All objects in the same segment are of the same data type. To find the data type of an object given its address, one takes the nine high-order bits of the address and uses them to index the data type table (called ST, for Segment Table). This table entry contains an encoding of the data type for objects in the corresponding segment:

| | |
|---|---|
| Bit 0 | 0 if atomic, 1 otherwise. |
| Bit 1 | 1 if list cells. |
| Bit 2 | 1 if fixnums. |
| Bit 3 | 1 if flonums. |
| Bit 4 | 1 if bignums. |
| Bit 5 | 1 if symbols. |
| Bit 6 | 1 if arrays (actually, array pointers; see below). |
| Bit 7 | 1 if value cells for symbols. |

| | |
|---|---|
| Bit 8 | 1 if number stack (one of bits 2-3 should also be set). |
| Bit 9 | is currently unused. |
| Bit 10 | 1 if memory exists, but is not used for data. |
| Bit 11 | 1 if memory does not exist. |
| Bit 12 | 1 if memory is pure (read-only). |
| Bit 13 | 1 if hunks. |
| Bits 14-17 | are currently unused. |
| Bits 18-35 | (the right half) contain a pointer to the symbol representing the data type, namely one of LIST, FIXNUM, etc. The symbol RANDOM is used for segments containing no standard MacLISP data objects. |

The encoding is redundant to take advantage of the PDP-10 instruction set and to optimize certain common operations. There is an instruction which can test selected bits in a half-word of an accumulator and skip if any are set. Thus, one can test for a number by testing bits 2, 3, and 4 together. Bit 0 (the sign bit) is 1 for list, hunk, and value cell segments (non-atoms) and 0 for all others (atoms). This saves an instruction when making the very common test for atom-ness, since one can use the skip-on-memory-sign instruction instead of having to fetch the table entry into an accumulator. The right half of a table entry contains a pointer to the symbol which the MacLISP function TYPEP is supposed to return for objects of that type. Thus, the TYPEP function need only extract the right half of a table entry; it does not have to test all the bits individually. Finally, the system arranges for all the symbols to which a table entry can point to be in consecutive memory locations in one symbol segment. Since these symbols have consecutive memory address, the right half of a table entry can be used to index dispatch tables by type. For example, the EQUAL function, which determines whether two LISP objects are isomorphic, first compares the data types of its two arguments; if the data types match, then it does an indexed jump, indexed by the right half of a Segment Table entry, to determine how to compare the two objects.

By way of contrast, let us briefly consider the storage convention formerly used by MacLISP. Memory was partitioned into several contiguous regions, not all of the same size. The lowest and highest addresses of each region were known (usually the low address of one region was one more than the highest address of the region below it). To determine the data type of a pointer it was necessary to compare the address to the addresses of all the boundaries of the regions. This was somewhat faster than the current table method if only one or two comparisons were needed (as in determining whether a pointer pointed to a number, since the number regions were contiguous), but slower in the general case; furthermore, there was no convenient way to dispatch on the data type. On the other hand, the table method requires space for the entire 512-word table, even if only a small number of segments are in use. (There is another 512-word table for use by the garbage collector, the GC Segment Table (GCST), which doubles this penalty.) The deciding advantage of the table method is that it permits dynamic expansion of the storage used for each kind of data. The region method requires all list cells, for example, to be in a contiguous region; once this region is fixed, there is no easy way to expand it. Under the table method, any currently unused segment can be pressed into service for list cells merely by changing its table entry. An additional bonus of the table scheme is that the space required for the instructions to do a type-check is small, and so it is often worth-while to compile such type-checks in-line in compiled code rather than calling a type-checking subroutine.

In practice new data segments are not allocated randomly, but from the top of memory down. As new pages of memory are needed they are acquired from the timesharing system and used for segments (on the ITS system, there are two segments per page). Compiled programs are loaded starting in low memory and working up; thus between the highest program loaded and the lowest data segment allocated there is a big hole in memory, which is eaten away from both ends as required. This hole has been whimsically named "the BIg Bag Of Pages" from which new ones are drawn as needed; hence the name "BIBOP" for the scheme. (The TOPS-10 timesharing system provided by DEC does not allow memory to be grown from the top down, but only from the bottom up. When running under this time-sharing system MacLISP has a fixed region for loading programs, and allocates new data segments from the bottom up.)

## DATA REPRESENTATIONS

List cells, as mentioned above, are represented as single words. The CAR pointer is in the left half of the word, and the CDR pointer in the right half.

Fixnums are represented as single words which contain the PDP-10 representation of the number. As explained more fully in reference 1, this representation permits arithmetic to be performed easily. If a pointer to a fixnum is in an accumulator, then any arithmetic instruction can access the value by indexing off that accumulator with a zero base address.

Flonums are represented as single words in a manner similar to fixnums.

Bignums each have a single word in a bignum segment. The left half of this word is all zeros or all ones, representing the sign of the number. This representation of the sign is compatible with that for fixnums and flonums; thus the sign of any number can be tested with the test-sign-of-memory instruction. (Bignums were formerly represented as list cells with special pointers in the CAR; this did not permit the compatibility of sign bits, and made it difficult to test for either numbers or lists.) The right half points to a list of positive fixnums, which represent the magnitude of the bignum, 35 bits per fixnum, least significant bits first in the list. A list is used instead of a contiguous block of storage for both ease of allocation and generality of use. The least significant bits come first in the list to ease the addition algorithm.

Symbols are quite complex objects. Each symbol has one word in a symbol segment and two words in another segment. The right half of the one word points to the symbol's property list, which is an ordinary list; the left half points to the two-word block. These two words in turn are laid out so:

| | | |
|---|---|---|
| bits | 0 | pointer to value cell |
| "args" property | | pointer to print name |

The "bits" have various specialized purposes. The value cell for the symbol is in a value cell segment. Notice that bits 13-17 of the first word are zero, specifying no indexing or indirection. This permits an instruction to indirect through this word to get the value of the symbol. Getting the address of the two-word block also takes an instruction; thus one can get the value of a symbol in two instructions. The "args" property is used by the MacLISP interpreter for checking the number of argument to a function (for symbols are also used to denote the names of functions). The print name is a list of fixnums containing the characters of the symbol's name, packed five ascii characters to the word.

The special symbol NIL is not represented in this manner. The address of NIL is zero. This allows a particularly fast check for NIL; one can use the jump-if-zero instruction. This is why accumulator 0 (which is also memory location 0) is reserved for NIL. Accumulator 0 normally contains zero itself; in this way taking CAR or CDR of NIL yields NIL. This allows one to follow a list by CDR pointers to a predetermined depth and not have to check at each step whether one has run off the end. (This trick was borrowed from InterLISP (ref. 2).) Most functions make special checks for NIL anyway, so this non-standard representation is not harmful. PRINT, for example, just checks for NIL specially and just outputs "NIL" without looking for a print name. NIL does have a property list, but it is not stored where it is in other symbols; the property list functions must check for NIL (which takes only one instruction anyway). NIL has no value cell, and always evaluates to NIL.

One might wonder why normal symbols are divided up into two parts, and why the value cell is not simply part of the two-word block. The answer is that once constructed the two-word block normally does not change, and so may be placed in read-only memory and shared between processes. If several MACSYMA processes are in use, this sharing may ease core requirements by tens of thousands of words.

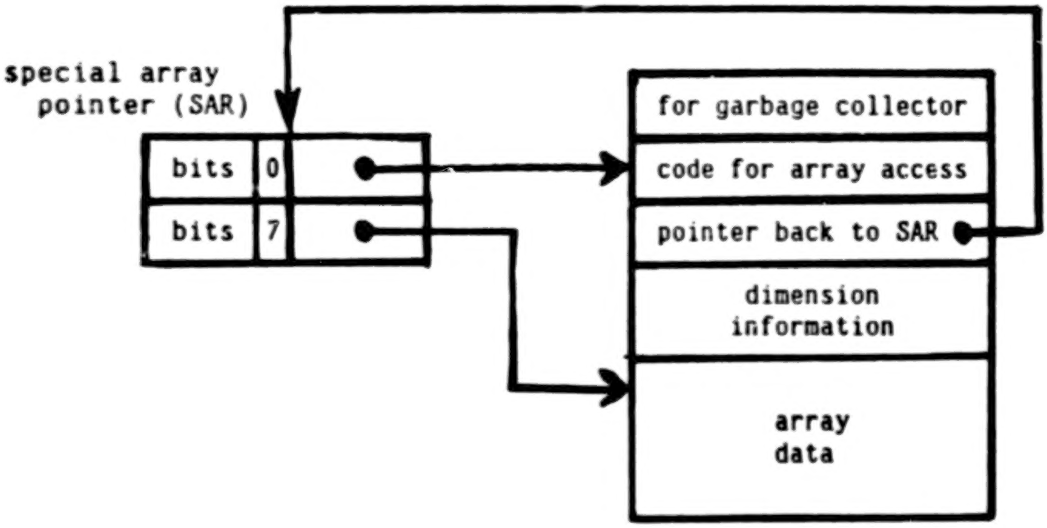
To save even more memory, symbols are not provided with value cells until necessary (most symbols are never actually given values). Instead, they are made to point to a "standard unbound" value cell, which is read-only and contains the marker specifying that no value is present. When an attempt is made to write into this value cell, the write is intercepted and a new value cell created for the symbol in question.

(Besides making parts of symbols read-only, MacLISP currently allows for read-only list cells, fixnums, flonums, and bignums. These are useful for constructing constant data objects which are referred to by compiled code but never modified, and for properties on property lists whose values are not expected to change (such as function definitions). In certain cases, such as the property-list modifying routines, checks are made for read-only objects, and such objects are copied into writable memory if necessary to carry out the operation. This copying causes the old read-only copy to be wasted from then on, but this is acceptable as such copying is seldom necessary in practice. This strategy may be contrasted to the approach of InterLISP (ref. 2), in which an entire page of memory is made writable if an attempt is made to modify any object on that page. This approach is more general than that of MacLISP, but in practice tends to reduce the sharing of pages among processes, increasing the load on the time-sharing system.)

Value cells, though not properly a MacLISP data type, are worthy of discussion. They are single words, containing a pointer in the right half and zero in the left half. This apparent waste of 18 bits is motivated by speed considerations. Compiled code often references the value cells of global variables. Since the left half of a value cell is zero, a test for NIL can be done with a single skip-if-memory-zero instruction; this is useful for switches. Furthermore, if a value cell is known to contain a list, the CAR or CDR can be taken in one instruction, using a half-word instruction with indirect addressing, because the index and indirection fields are zero, without having to fetch the value into an accumulator first. Similarly, if a value cell contains a number, the sign can be tested and the value (except for bignums) accessed by using indirect addressing. (It should be noted that compiled code does not keep local variable values in value cells, but uses even more clever techniques involving stacks.)

Arrays have a complicated representation because they can be of arbitrary size, and must be allocated as a contiguous block for efficient indexing. The solution chosen is to split it into two parts: a Special ARray cell (called SAR, not SAC, for some reason) in an array segment, and the block of data. The data itself is kept just below the hole in memory, floating above loaded programs. When new programs are loaded, the array data is shuffled upward in memory, and the special array pointers are updated. Similarly, when allocating a new array or reclaiming an old one it may be necessary to shuffle the array data.

The special array pointer is two words:



A complete discussion of the SAR contents and array access methods is beyond the scope of this paper. Notice, however, that the indirection and index fields are chosen to be 0 and 7 for the two SAR words. The first admits an indirection for calling the array as if it were a function, according to MacLISP convention; the second allows indexing off accumulator 7 for accessing the data from compiled code. See reference 1 for a fuller treatment of this.

Hunks are like list cells, but consist of several contiguous words. They are always a power of two in size, for convenience of allocation. Hunks of sizes other than powers of two are created by allocating a hunk of a size just big enough, and then marking some of the halfwords as being unused by filling them with a -1 pointer (actually 777777). This was chosen because it never points to a data object, and because it is easily generated with instructions that set half- or full-words to all ones. It is time-consuming to determine the actual size of a hunk, since one must count the number of unused halfwords, but then hunks were created as an experimental space-saving representation with properties somewhere between those of lists and arrays.

## GARBAGE COLLECTION

Every so often there comes a point when all the space currently existing for data objects has been allocated. At this point there are two alternatives:
[1] allocate a new segment for data objects of the type needed.
[2] attempt to reclaim space used by data objects which are no longer needed (by the process of garbage collection).
A study by Conrad indicates that the best strategy is to do [2] only if [1] fails because one's address space (256K words, in this case) is completely allocated, PROVIDED that one has the facility to compact one's data storage and de-allocate

segments. (Ref. 3) Since MacLISP currently hasn't the ability to de-allocate segments ("once a fixnum, always a fixnum"), this strategy must be modified. One must be cautious about allocating a new segment, since the allocation cannot be undone; thus MacLISP tries garbage collection first unless explicitly told otherwise by the programmer, and then allocates a new segment if garbage collection fails to reclaim enough space for the required data type.

Suppose, for example, that it is necessary to allocate a new list cell. The CONS function checks the freelist for the data type "list cell"; if the freelist is not empty, then the first cell on that list is used. (There is a freelist for each data type, which consists of all the currently unused objects in all the segments for that data type, strung together such that each object points to the next. This can be done even for objects which ordinarily do not contain pointers, such as fixnums and flonums, since those objects are large enough to contain at least a single pointer. There is a set of fixed locations, one for each data type, which contain pointers to the first cells on the respective freelists.)

If, in our example, the list cell freelist is empty, then the garbage collector is invoked. Controlled by user-settable parameters, the garbage collector may decide simply to allocate a new list segment (which involves getting a new memory page from the time-sharing system, altering the Segment Table, and adding the newly allocated objects to the freelist). If it decides not to do this, or if the attempt fails for any reason, then the actual garbage collection process is undertaken. This involves finding all the data objects which are accessible to the user program. An object is accessible if it is pointed to by compiled code, if pointed to by a global variable or internal pointer register (such as accumulators 1-5), or if pointed to by another accessible object. Notice that this definition is recursive, and so requires a recursive searching of all the data objects to determine which are accessible. This searching is known as the mark phase of the garbage collector.

Associated with each data object is a "mark bit" for use by the garbage collector. As the garbage collector locates each accessible object, it sets that object's mark bit. For list cells, fixnums, flonums, bignums, and hunks, these bits are stored in a part of memory unrelated to the memory occupied by the data objects themselves. For each 512-word segment there is a "bit block" of 16 words, each holding 32 mark bits. The location of the bit block is found by using the top 9 bits of the address of the data object to index the GC Segment Table. (Bit blocks themselves are allocated in special "bit block" segments; thus bit blocks are treated internally as yet another data type. Occasionally the obscure error message "GLEEP - OUT OF BIT BLOCKS" is printed by LISP in the highly infrequent situation where it cannot allocate a new bit block after allocating a new segment which needs a bit block.) No bit blocks are needed for symbols and special array pointers. Recall that the left half of a symbol word points to a two-word block. Since such a two-word block is always at an even address, the low bit of the pointer to it is normally zero. This bit is used during garbage collection as the mark bit for that symbol. Special array pointers have room in them for a variety of bits, and one of them is used as a mark bit. Value cells are only reclaimed when the symbol pointing to them is reclaimed (and not even then, if compiled code points to the value cell, which fact is indicated by a bit in the two-word symbol block pointing to the value cell), and so they require no mark bits.

To aid the garbage collector in the mark phase, the GCST contains some bits which also encode the data type redundantly, in a form useful to the marking routine. The bits indicate whether the object must be marked, and if so the method of marking; they also indicate how many pointers to other objects are contained in the object now being marked.

After recursively locating and marking all accessible cells, the garbage collector then performs a sweep phase, in which every data object is examined, and those which have not been marked are added to the appropriate freelist. To aid the sweep phase, each GCST entry has a field by which all entries for segments of the same data type are linked together in a list. In this way the garbage collector does not need to scan the entire segment table looking for entries for each type. For each segment, the garbage collector examines each data object in the segment and its mark bit, and adds the object to the appropriate freelist if the mark bit is not set. For symbols and arrays it also resets the mark bit at this time. (Bit blocks are reset at the beginning of the mark phase.)

If, in our example, the garbage collection process has not reclaimed enough list cells (as determined by another programmer-specified parameter), then it will try to allocate one or more new list cell segments. If, however, this causes the total number of list cells to exceed yet another programmer-specified parameter, then a "user interrupt" is signaled, and a function written by the programmer steps in. In MACSYMA, this function is the one that typically informs you:

```
YOU HAVE RUN OUT OF LIST SPACE.
DO YOU WANT MORE?
TYPE ALL; NONE; A LEVEL-NO. OR THE NAME OF A SPACE.
```

The reason for all these parameters is the necessary caution described above; if all the available segments get allocated as list cell segments (which can easily happen due to intermediate expression swell, for example), then they cannot be used for anything else, including compiled code. This is why, in MACSYMA, if you use up too much list space, you can't load up DEFINT thereafter!

Array data (as opposed to the SAR objects) is handled by a special routine that knows how to shuffle them up and down in core as necessary. When a new array is allocated, the garbage collector has the same decision to make as to whether to allocate more memory or attempt to reclaim unused arrays. The decision here is less critical, since memory allocated for arrays CAN be de-allocated, and so no programmer-specified parameters are used. Array data only goes away when the corresponding SAR is reclaimed by the normal garbage collection process (or when the array is explicitly killed by the user, using the *REARRAY function).

For the interested reader, the format of a GCST entry is shown here:

| | |
|---|---|
| Bit 0 | 1 if data objects in this segment must be marked. |
| Bit 1 | 1 if this segment contains value cells. |
| Bit 2 | 1 if symbols. |
| Bit 3 | 1 if special array pointers. |
| Bit 4 | 1 if the right half of this data object contains a pointer (true of list, bignum, and hunk data objects). |
| Bit 5 | 1 if the left half of this data object contains a pointer (true of list and hunk objects -- note that symbols and special array pointers get special treatment). It is always true that bit 4 is set if this one is. |
| Bit 6 | 1 if hunks (in this case, the ST entry is used to determine the size of the hunk). |
| Bits 7-12 | are unused. |
| Bits 13-21 | contain the index into GCST of the next entry with the same data type, or zero if this is the last such entry. (Segment 0 never contains data objects, except NIL, which is treated specially anyway.) |

Bits 22-35 contain the high 14 bits of the address of the bit block for this segment, if any.

Since bit blocks are 16 words long, the low four bits of the address of such a bit block are always zero. Thus the GCST entry only needs to contain the high 14 bits of the address. These 14 bits are right-adjusted in the GCST entry for the convenience of a clever, tightly-coded marking algorithm. This algorithm works roughly as follows:

[a] Shift the address of the data object to be marked right by 9 bits, putting the low 9 bits into the next accumulator.
[b] Use the high 9 address bits to fetch a GCST entry into the accumulator holding the high 9 address bits, skipping on the sign bit (whether to mark or not).
[c] Test bits 1, 2, 3 (special treatment), skipping if none are set.
[d] Shift the two accumulators left by 4 bits. This brings four of the low 9 address bits back into the first accumulator, which together with 14 bits from the GCST entry yield the address of a word in the bit block. The 5 bits remaining in the second accumulator indicate the bit within the word to use as the mark bit. Finally, bit 4 is brought into the sign bit of the first accumulator.
[e] Rotate the second accumulator, bringing the 5 bits to the low end.
[f] Indexing off the first accumulator, fetch the word of mark bits.
[g] Set a mark bit in the word, skipping if it was not already marked. (If this doesn't skip, then we exit the marking algorithm. It is not necessary to store back the word of mark bits.) The bit is selected by indexing off the second accumulator into a table of words, each with one bit set.
[h] Store back the word of mark bits.
[i] Test the sign bit of the first accumulator (bit 4 of the GCST entry), jumping to the exit if not set.
[j] If bit 1 is set (bit 5 of the GCST entry), recursively mark the pointer in the left half. If bit 2 is set (bit 6 of the GCST entry), mark all the pointers in the hunk.
[k] Iteratively mark the pointer in the right half.

I have taken the trouble to outline these steps carefully because most of them are single PDP-10 instructions, carefully designed to perform two or three useful operations simultaneously. The point is that the careful design of tables and the use of redundant encoding can greatly increase the speed of critical inner loops. (It should also be mentioned that such careful thought about design is usually warranted only for critical inner loops!) I should also mention that most of the constants which have been mentioned in this paper (bit numbers, sizes of segments, and so on) are represented symbolically in the text of the MacLISP code; one can change the size of a segment by changing a single definition, and the sizes of fields in GCST entries, positions of bits, and so on will be adjusted by assembly-time computations. I have used numbers in this paper only for concreteness.

For certain spaces the mark bits are actually used in the inverted sense: 1 means not marked, and 0 means marked. This allows the sweep loop to test for an entire block of 32. words all being marked by testing for a zero word of mark bits; the loop can then just skip over the block, and avoid testing the individual bits. The test for a zero word is done while moving the word into an accumulator, which has to be done anyway, and so is essentially free.

## THE ADDRESS SPACE PROBLEM

One of the difficulties currently facing MacLISP is the "limited" address space provided by the PDP-10. The architecture of the machine inherently limits addresses to 18 bits; hence a single program cannot address more than 256K words of memory. Combined with the fact that MacLISP does not presently allow for de-allocation of data segments (or of loaded compiled code, for that matter), this severely limits the use of memory. Some MACSYMA problems, for example, would require much more than 256K of programs and list data to solve; others require less than 256K at any one time, but cannot be run because of the de-allocation difficulty.

It is fairly clear that completely solving the de-allocation problem would be more trouble than it is worth, and would not stave off the fundamental difficulty indefinitely. As both MACSYMA problems and MACSYMA itself grow in size, we will feel more and more the "address space crunch". The only general way to solve this problem is to arrange for a bigger address space.

There are three solutions which are presently at all realistic. Two involve continued use of the PDP-10 architecture, but modified in several ways to allow programs to access more memory. These modifications may or may not be made available by DEC, and may or may not be retrofittable to the MACSYMA Consortium KL10 processor. The difference between the two schemes involves the decision as to whether MacLISP data pointers should still fit into 18 bits. If not, there is immediately a factor-of-two memory penalty, since list cells must be two words instead of one. However, there are also some technical advantages to such an arrangement, as well as the obvious advantage that list space can become bigger than 256K. If pointers are kept to 18 bits, then all LISP data must fit in 256K, but any amount of compiled code and any number of arrays could be loaded. Both of these schemes have been worked out on paper to a great extent by Guy L. Steele Jr. and Jon L. White, to compare their merits and to prepare for the possibility that one of them may be needed. Either scheme would require a good deal of work (at least one to two man-years) to implement fully in both the interpreter and the compiler.

The third solution involves moving to another machine architecture altogether. This leaves open the choice of machine. Few commercially available machines are as conducive to the support of LISP as the PDP-10, and it probably would not be practical to undertake a completely new implementation. MacLISP does presently run on Multics (on a Honeywell 6180 processor), but is rather slow, and the Multics system is expensive and not widely available. The best bet in this direction seems to be the LISP machine, designed by Richard Greenblatt, Tom Knight, et al. at the MIT Artificial Intelligence Laboratory. The prototype machine has been working for a number of months now, and the basic software is beginning to show signs of life. It is not inconceivable that MACSYMA may be run experimentally on it by summer 1977. The LISP machine has a 23-bit address space, and makes more efficient use of its memory than even the PDP-10. However, although it is much less expensive than a KL10, it is not designed for time-sharing.

The PDP-10 implementation of MacLISP and of MACSYMA will certainly be useful for at least the next five to ten years. After that, only time can tell.

## SUMMARY

MacLISP is designed to be an efficient, high-level systems programming language, rather than primarily an applications programming language. Its internal organization is a carefully chosen balance between useful generality and special-case efficiency tricks. A thoughtful choice of data and table representations can exploit the architecture of the host machine to gain speed in critical places without great loss of generality. The use of symbolic assembly parameters can avoid tying the system to a single rigid format. The greatest effort has been expended on speeding up type-checking, access to values in global variables, and garbage collection, since these are among the most frequent of LISP operations. The address space crunch may eventually force yet another redesign if the PDP-10 architecture is retained.

## REFERENCES

1. Steele, Guy L. Jr.: "Fast Arithmetic in MacLISP." Proceedings of the MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper No. 22 of this compilation.)

2. Teitelman, Warren: InterLISP Reference Manual. Revised edition. Xerox Palo Alto Research Center (Palo Alto, 1975).

3. Conrad, William R.: A compactifying garbage collector for ECL's non-homogeneous heap. Technical Report 2-74. Center for Research in Computing Technology, Harvard U. (Cambridge, February 1974).

# FAST ARITHMETIC IN MACLISP

Guy Lewis Steele Jr.
Massachusetts Institute of Technology
Laboratory for Computer Science
(formerly Project MAC)

## ABSTRACT

MacLISP provides a compiler which produces numerical code competitive in speed with some FORTRAN implementations and yet compatible with the rest of the MacLISP system. All numerical programs can be run under the MacLISP interpreter. Additional declarations to the compiler specify type information which allows the generation of optimized numerical code which generally does not require the garbage collection of temporary numerical results. Array accesses are almost as fast as in FORTRAN, and permit the use of dynamically allocated arrays of varying dimensions. Here we discuss the implementation decisions regarding user interface, data representations, and interfacing conventions which allow the generation of fast numerical LISP code.

## INTRODUCTION

For several years now MacLISP has supported a compiler which produces extremely good numerical code. Measurements made by Fateman indicate that the generated code is competitive with FORTRAN. (Ref. 1) Expressing such numerical code does not require the use of special numerical language embedded within LISP, in the manner that some higher-level languages allow the user to write machine code in the middle of a program. Rather, all numerical programs are completely compatible with the MacLISP interpreter. The compiler processes the interpreter definitions along with additional numerical declarations. These declarations are not required; omitting them merely results in slower compiled code. For convenience, special numeric functions are provided which carry implicit declared type information (such as + and +$ for integer and floating point addition, as opposed to PLUS), but the user need not use them to get optimized numerical code.

## CHANGES TO THE MACLISP LANGUAGE

The primary change to the MacLISP language, as seen by the user, was the creation of numerical declarations for use by the compiler. A general compiler declaration mechanism was already a part of the language, so adding the numerical declarations was not difficult. This mechanism involves writing a MacLISP expression beginning with the word DECLARE and followed by various declarations. Declarations may be global or local. Global declarations are written by themselves in a file, and affect all following functions; local declarations are written within the text of a MacLISP function, and affect only the scope of the construct they are written within.

The simplest new declarations are statements of the types of variables. Recall that MacLISP has three basic numeric types: fixnum, flonum, and bignum. These are (respectively) single-precision integers, single-precision floating-point numbers, and arbitrary-precision integers. Only the first two types can be operated on directly by hardware instructions, and so they are the only types of interest to the compiler. An example of a variable declaration:

```
(DECLARE (FIXNUM I J K)          ;single-precision integers
         (FLONUM A B FOO ZAP)    ;single-precision reals
         (NOTYPE SNURF QUUX))    ;no specific type
```

If a variable is always numeric but sometimes may hold bignums, it must be declared NOTYPE. The default assumption is that a variable is NOTYPE (that is, may contain any MacLISP data object); NOTYPE declarations are primarily useful to undo previous numeric declarations.

The types of the arguments and returned values of functions may be similarly declared:

```
(DECLARE (FLONUM (CUBE-ROOT FLONUM)
                 (INTEGER-POWER-OF-REAL FLONUM FIXNUM))
         (FIXNUM (FIBONACCI FIXNUM)
                 (LENGTH-OF-LIST NOTYPE))
         (NOTYPE (BETWEEN-ZERO-AND-ONE-PREDICATE FLONUM)))
```

This declaration specifies that CUBE-ROOT takes a FLONUM argument and delivers a FLONUM result, that INTEGER-POWER-OF-REAL takes a FLONUM and a FIXNUM and delivers a FLONUM, and so on. The types of the arguments could also be specified by using a local declaration:

```
(DECLARE (FLONUM (CUBE-ROOT)))  ;global declaration

(DEFUN CUBE-ROOT (X)
       (DECLARE (FLONUM X))     ;local declaration
       (EXPT X .333333333))
```

The result type must be specified by a global declaration, however, and declaring the argument types globally also can help the compiler to produce better code for functions which call the declared function.

Arrays may also be declared globally to the compiler. MacLISP arrays come in three types, which are essentially FIXNUM, FLONUM, and NOTYPE. (There are other types also, but these do not concern us here.) The ARRAY* declaration takes a subdeclaration specifying the array type; the subdeclaration in turn specifies the names of arrays and their dimensions. An example:

```
(DECLARE (ARRAY* (FIXNUM TUPLE 1 TABLE 2)
                 (FLONUM VECTOR 1 MATRIX 2)))
```

This declares TUPLE and VECTOR to be one-dimensional arrays, and TABLE and MATRIX to be a two-dimensional arrays. (MacLISP arrays may have up to five dimensions.) If the values of the dimensions are also known ahead of time, a slightly different form may be used:

```
(DECLARE (ARRAY* (FIXNUM (TUPLE 43) (TABLE 3 5))
                 (FLONUM (VECTOR 3) (MATRIX ? 17))))
```

This declares TUPLE to be of length 43, TABLE to be 3 by 5, and MATRIX to have 17 columns and an unknown number of rows. Note that "?" can be used to denote an unknown dimension value; even partial dimension information can help the compiler to optimize array accesses.

The user can write arithmetic code using the traditional names PLUS, DIFFERENCE, TIMES, and QUOTIENT; these functions work on any kinds of numbers, even bignums, and admit mixed-mode arithmetic. In the presence of type declarations, the compiler may be able to deduce that the arguments are always flonums, for example, and produce hardware instructions for floating-point arithmetic. The user can also use the FIXSW and FLOSW declarations to tell the compiler that such "generic" arithmetic will always involve only fixnums or only flonums.

As a convenience to the user, however, several versions of the common arithmetic functions are provided:

| generic | fixnum only | flonum only |
|---|---|---|
| PLUS | + | +$ |
| DIFFERENCE | - | -$ |
| TIMES | * | *$ |
| QUOTIENT | // | //$ |
| REMAINDER | \ | |
| GCD | \\ | |
| GREATERP | > | > |
| LESSP | < | < |
| EQUAL | = | = |
| EXPT | ^ | ^$ (fixnum exponent) |

(The division functions are written as "//" instead of "/" because "/" is a MacLISP escape character.) The functions in the last two columns are completely equivalent to those in the first column, except that they convey additional type information about their arguments and results. (An exception is that the fixnum-only functions do not check for overflow, so in a situation where, for example, 100000000 and 100000000 were multiplied together, TIMES would produce a bignum, whereas * would overflow and produce a not-very-meaningful fixnum. The flonum-only functions do not check for overflow either, whereas the generic functions give an error for overflow, and either an error or zero for underflow.)

## CHANGES TO THE MACLISP IMPLEMENTATION

In order that the arithmetic machine instructions might be used directly on MacLISP numeric data objects, it was necessary to modify MacLISP to use a uniform representation for fixnums and flonums. Before the fast-arithmetic scheme was implemented, MacLISP, like many other LISP systems, used two representations for single-precision integers. One represented the integer as a pointer to a machine word containing the value, in the same manner as floating-point numbers were represented. The other encoded the value into the pointer itself, using pointer values which were otherwise worthless because they pointed at code instead of data objects. The motivation behind the earlier dual representation was to avoid allocating storage for small integer values, which are frequently used. (InterLISP has for several years "open-compiled" arithmetic functions as single machine instructions. (Ref. 2) Unfortunately, it still has a dual representation for integers; as a result, before adding two numbers it must call a routine which determines at run-time the representation of each number and converts each into a full machine word representation. Compiled InterLISP code also calls a similar routine for floating-point numbers, not because of multiple representations, but in order to perform error-checking as completely as the interpreter does. This run-time checking destroys any advantage gained by open-compiling the arithmetic instructions.)

The pointer encoding was removed from MacLISP for the fast-arithmetic scheme, and all numbers are now uniformly encoded as pointers to full machine words which contain the machine representations of the values. In order to avoid allocating storage for frequently used small integers, there are several hundred words of memory containing consecutive small integer values, and small integers are created by making a pointer to one of these standard locations, rather than allocating a new word for each use of a small integer. (MacLISP does not allow the words used to contain numbers to be modified in the way InterLISP allows using the SETN primitive (ref. 2), so there is no difficulty in sharing such words. In fact, these small integer locations are even shared among all the MacLISP processes in the time-sharing system by making them read-only.)

While arithmetic on bignums cannot be compiled as standard arithmetic machine instructions, their representation has been chosen to permit sign tests to be open-compiled. A bignum is a pointer to a word which has the sign of the bignum in the sign bit (and in fact the entire left half), and a pointer to a list of fixnums (which represent the magnitude) in the right half. Thus all numbers are pointers to words which contain the sign of the number in the sign bit, and such functions as MINUSP can always be compiled as single machine instructions.

In order to preserve the uniformity of the function-calling interface, it was decided that all arguments to functions must be valid MacLISP data objects. On the other hand, it is not desirable to have to "number cons" out of free storage, with the garbage collection overhead that implies, in order to pass numbers to functions. The solution used was to introduce two extra pushdown lists (stacks) called the fixnum and flonum pdls. The storage in these pdls appear to have fixnum or flonum data type, but they are allocated as stacks rather than as garbage-collected heaps. These stacks can be used to hold temporary numerical values and the values of PROG variables which have been declared to be numeric, but they can also be used to allocate pseudo-data objects compatible with MacLISP's standard number representation. A pointer to a fixnum pdl location is indistinguishable from an ordinary fixnum for most purposes; it is a pointer to a full machine word containing the numeric value. A typical code sequence resulting from compiling (FOO (+ A 5)) is:

```
;assume accumulator 1 has the pointer value of A in it
        MOVE 7,(1)      ;get the machine word for A into accumulator 7
        ADDI 7,5        ;add 5 to the machine word
        PUSH FXP,7      ;push resulting word into fixnum pdl
        MOVEI 1,(FXP)   ;copy fxp pointer into argument accumulator 1
        CALL 1,FOO      ;call foo
        SUB FXP,[1,,1]  ;remove pushed word from fixnum pdl
```

To the function FOO the pointer passed in accumulator 1 has the precise format of a MacLISP integer: a pointer to a machine word containing the integer value. Note that the value of the variable A may itself have been such a "pdl number"; the MOVE instruction would move the machine word value into accumulator 7 whether it was a pdl number or an ordinary fixnum.

One of the difficulties of using stack-allocated numbers is that they have a definite lifetime; on return from the function they are passed to, they are de-allocated and no longer exist. By the time they are de-allocated, there must be no more pointers to that word accessible to the user program, or else subsequent references might see a wrong value because the pdl word was re-allocated for some other purpose.

To overcome this difficulty the notion of safety was developed. A copy of a pointer is safe if it can be guaranteed that the copy will become inaccessible before what it points to is de-allocated if the pointer in fact points to a pdl number. Alternatively, a use for a pointer is safe if that use doesn't require a safe pointer. The fast-arithmetic compiler does some complex analysis to determine what situations are safe. Some standard conventions for safety:

[1] A pointer in a global (special) variable may have an indefinite lifetime, and so putting a pointer in a global variable is unsafe. It follows that such a variable may not contain a pointer to a pdl number, since we cannot guarantee such a pointer to be safe. Consequently, any pointer actually obtained from a global variable is safe.

[2] Consing a pointer into a list cell (or using RPLACA to put a pointer into an existing list call) is similarly unsafe. Pointers actually occurring in list structure must therefore be guaranteed safe.

[3] It is not possible to return a pdl number as the value of a function, because there would be no return to the code to de-allocate it. Therefore returning a pointer from a function is unsafe, and all pointers actually returned from functions are safe.

[4] Passing a pointer as an argument to a function is safe; therefore pdl numbers (unsafe pointers) may be passed as arguments to functions. All function arguments are thus potentially unsafe. They may be passed on down to other called functions, but may not be returned or otherwise used as if they were safe.

[5] Pdl numbers may be pointed to by ordinary compiled local variables. Such local variables may or may not have unsafe values, depending on where the values came from. The compiler must guarantee that when the value of a local variable is used either the value is safe or the use is safe.

Suppose we wrote a function such as:

```
(DEFUN ZAP (A) (CONS A 'FOO))
```

We are putting the argument A into a list cell (an unsafe use), but the argument A is also (potentially) unsafe. In this situation the compiled code must create a safe copy of the unsafe pointer. The compiled code therefore uses a routine PDLNMK ("pdl number make") which checks for a pdl number and makes a copy by doing a number cons if necessary. That is, if the pointer given to PDLNMK is already safe, it is returned as is; but if it is unsafe, a safe copy is made with the same value. The compiled code for ZAP would look like this:

```
MOVEI 2,'FOO     ;put constant "foo" in accumulator 2
JSP T,PDLNMK     ;make sure accumulator 1 has a safe pointer
JCALL 2,CONS     ;call CONS
```

If A is not a pdl number, PDLNMK does nothing; but if it is, PDLNMK replaces the pointer in accumulator 1 with a freshly allocated fixnum with the same value as the pdl number. In this way a safe value will be passed to the CONS function. (The convention about function arguments being potentially unsafe has an exception in CONS, so that CONS itself need not always perform PDLNMK on its arguments. The compiler knows about this exception, and guarantees that anyone who calls CONS will provide safe arguments. In practice, arguments passed to CONS often can be guaranteed safe by compile-time analysis, and it saves time not to have CONS use PDLNMK.)

Notice that one consequence of the use of PDLNMK is that two numbers which are apparently EQ (i.e. the same pointer) may not be if the compiled code has to make a copy. For example, consider this code:

```
(DEFUN LOSE (X)
       (SETQ G X)
       (EQ X G))
```

The result of the EQ test could be NIL, even though the global variable G apparently is assigned the same pointer as was passed to LOSE as an argument. If an unsafe pointer is passed to LOSE, G will receive a safe copy of that value, which will not be the same pointer, and so the EQ test will fail. (This is another reason why MacLISP does not have a SETN primitive; since the compiler can make copies of a number without warning, conceivably SETN might modify one copy of a number but not the other, with anomalous results.)

Recall that one unsafe use of a pointer is returning it as the value of a function. We would like for numeric code not to ever have to "number cons", but we cannot return a pdl number from a function. The solution to this dilemma is to allow numeric-valued functions to have two entry points. One is the standard MacLISP entry point, and is compatible with the standard MacLISP calling sequence; calling the function there will produce a MacLISP pointer value, which will involve a number cons if the value is in fact numeric. The other is a special entry point which is non-standard, and can only be used by compiled code which knows that the called function is numeric-valued. Entering a numeric function there will deliver a machine word in accumulator 7 instead of the standard pointer in accumulator 1.

In order to use this special calling sequence, both the called function and the calling function must be compiled with declarations specifying that the called function is numeric-valued. The compiler will then compile the called function to have two entry points, and the calling function to use the non-standard numeric entry point.

The entry points are actually implemented as two consecutive locations at the beginning of the function. The first is the standard entry point; it merely pushes the address of a special routine FIX1 (or FLOAT1, for a flonum-valued function) onto the stack, and then falls into the non-standard entry point. The function then always produces a machine number in accumulator 7. If the function is called at the numeric entry point, it will deliver its value as a machine word. If called at the standard entry point, then on delivering the machine word it will "return" to FIX1, which performs a "number cons" on the machine word, producing a normal fixnum (or FLOAT1, which produces a flonum), and then returns to the caller.

As an example, here are two functions with appropriate declarations:

```
(DECLARE (FLONUM (DISC FLONUM FLONUM FLONUM)))

(DEFUN DISC (A B C)
       (-$ (*$ B B) (*$ 4.0 A C)))

(DEFUN QUAD (A B C)
       (PROG (D)
             (DECLARE (FLONUM D))
             (SETQ D (DISC A B C))
             (COND ((MINUSP D) (RETURN (ERROR)))
                        (T (RETURN (//$ (-$ (SQRT D) B)
                                        (*$ A 2.0)))))))
```

The code produced would look like this:

```
        DISC:   PUSH P,[FLOAT1] ;for normal entry, push address of FLOAT1
                MOVE 7,(2)      ;numeric entry point; get machine word for B
                FMPR 7,7        ;floating multiply B by itself
                MOVSI 10,(4.0)  ;get 4.0 in accumulator 10
                FMPR 10,(1)     ;floating multiply by A
                FMPR 10,(3)     ;floating multiply by C
                FSBR 7,10       ;floating subtract ac 10 from ac 7
                POPJ P,         ;machine word result is in ac 7
```

Notice that DISC does no number consing at all if called at the numeric entry point. It does all arithmetic in the accumulators, and returns a machine word as its result. The code is remarkably compact, of the kind one ordinarily expects from a FORTRAN compiler.

```
        QUAD:   PUSH P,1        ;save A, B, and C on the stack
                PUSH P,2        ; to preserve them across the
                PUSH P,3        ; call to DISC
                NCALL 3,DISC    ;call DISC with the same arguments
                PUSH FLP,7      ;push the result onto flonum pdl
                JUMPGE 7,G0003  ;jump if value non-negative
                MOVEI T,0
                CALL 16,ERROR   ;call the ERROR routine
                JRST G0005      ;go to G0005
        G0003:  MOVEI 1,(FLP)   ;get a pointer into flonum pdl
                NCALL 1,SQRT    ;call SQRT with that pointer
                FSBR 7,@-1(P)   ;floating subtract machine value of B
                MOVE 10,@-2(P)  ;fetch machine word value of A
                FSC 10,1        ;multiply by 2.0 (using "floating scale")
                FDVR 7,10       ;divide ac 7 by ac 10
                JSP T,FLCONS    ;perform a flonum cons
        G0005:  SUB P,[3,,3]    ;clean up the stacks
                SUB FLP,[1,,1]
                POPJ P,         ;return pointer value in accumulator 1
```

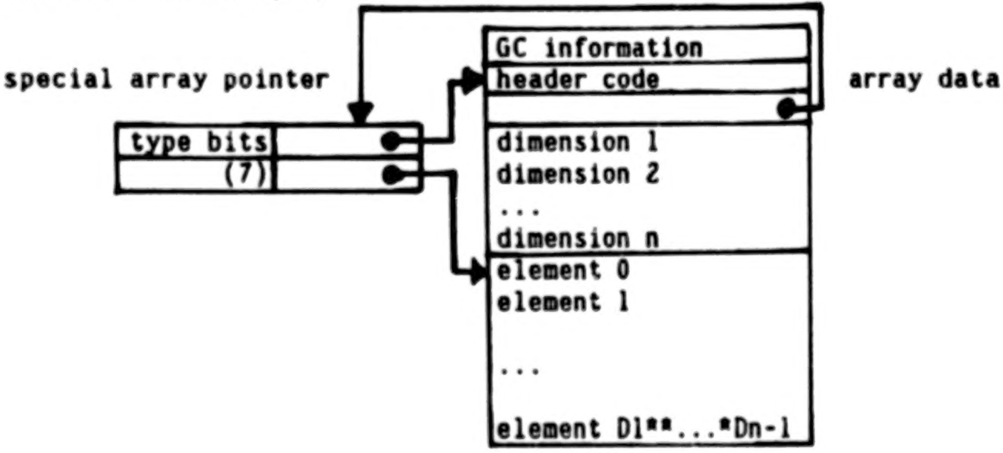
There are several points to note about QUAD:
(1) It was not declared to be numeric-valued. As a result, when returning a number it must do a number cons. Moreover, it does not have a numeric entry point.
(2) Because DISC was declared to be numeric-valued, QUAD uses NCALL instead of CALL to invoke it; NCALL enters at the numeric entry point. The result of DISC is expected in accumulator 7. Since QUAD needs to use this result to pass to SQRT, it makes a pdl number out of this machine word. In this way function values can be made into pdl numbers after all -- but by the caller rather than the called function.
(3) As an aside, the compiler makes some other neat optimizations. It uses a JUMPGE instruction for MINUSP, because the value to be tested is in an accumulator anyway. It takes advantage of the address arithmetic of the PDP-10 to fetch machine words pointed to by pointers on the stack in one instruction. It knows how to use several accumulators for arithmetic, and to arrange for the result to end up

in the correct accumulator. It expresses the multiplication by 2.0 as a "floating scale" instruction, which is faster than the multiplication instruction if one operand is a floating-point power of two.

The representation of arrays in MacLISP was carefully redesigned to allow fast access to them by compiled code, again taking advantage of the powerful address arithmetic of the PDP-10. There are essentially two kinds of arrays: s-expression arrays, whose components may be any safe pointers, and numeric arrays, whose components must be all fixnum machine words or all flonum machine words.

The MacLISP ARRAY data type is a pointer to a double word (the "special array pointer") which in turn points to the array data. The reason for this is that the pointer must point to a fixed place (as all MacLISP pointers must), but the actual array data may have to be shifted around by the garbage collector to accommodate new storage requests, because arrays are not of a uniform size. When the garbage collector moves the array data, it updates the the contents of the special array pointer, but the special array pointer itself may remain in a fixed place.

In exchange for the flexibility of dynamically allocated arrays, however, one pays the price of always accessing the array data indirectly through the special array pointer. This cost is alleviated by taking advantage of addressing arithmetic. The second word of each special array pointer points to the array data, which is arranged linearly in row-major order; this second word furthermore specifies indexing by accumulator 7.



Compiled code can access a numeric array datum by calculating the linear subscript value in accumulator 7 and then using an indirect fetch through the second word of the special array pointer for the array. The linear subscript value is of course calculated as

$$(\ \ldots\ (J1 * D2 + J2) * D3 + J3\ \ldots\ ) * Dn + Jn$$

where the Ni are the dimensions of the array and the Ji are the actual subscripts. For example, suppose that accumulator 1 contains a pointer to a 3 by 5 by 13 fixnum array, and that accumulators 2, 3, and 4 contain fixnum subscripts for that array. Then to fetch the desired datum this code would be used:

```
MOVE 7,(2)        ;fetch first subscript into ac 7
IMULI 7,5         ;multiply by 5 (second dimension)
ADD 7,(3)         ;add in second subscript
IMULI 7,13        ;multiply by 13 (third dimension)
```

```
        ADD 7,(4)       ;add in third subscript
        MOVE 7,@1(1)    ;fetch indirect through special array pointer
```

If the number of dimensions of the array has been declared to the compiler but not the values of the dimensions, the compiler arranges to fetch the dimension values at run time. This is easy because the array is arranged so that negative subscript values fetch the dimension information. (The LISP user is not supposed to use this fact, but only compiled code.) The same example for a three-dimensional array of arbitrary dimensions might look like this:

```
        MOVE 10,(2)     ;fetch first subscript into ac 10
        MOVNI 7,2       ;put -2 into ac 7
        IMULI 10,@1(1)  ;multiply by second dimension
        ADD 7,(3)       ;add in second subscript
        MOVNI 7,1       ;put -1 into ac 7
        IMULI 10,@1(1)  ;multiply by third dimension
        ADD 10,(4)      ;add in third subscript
        MOVE 7,10       ;move into ac 7 for subscripting
        MOVE 7,@1(1)    ;fetch indirect through special array pointer
```

The code is a little longer than before, but will work for any three-dimensional array. In general, the compiler tries to minimize subscript computations. If the exact dimensions are declared, or if some of the subscripts are constant, the compiler will do part or all of the subscript calculations at compile time.

For s-expression arrays, the pointer data are stored two per word, with elements having even linear subscripts in the left half of a word and the succeeding odd subscripted elements in the right half of the word. The compiler must generate code to test the parity of the linear subscript and fetch the correct half-word. Suppose that a pointer to a one-dimensional array is in accumulator 1, and a fixnum subscript is in accumulator 2. Then the following code would be generated:

```
        MOVE 7,(2)      ;fetch subscript into ac 7
        ROT 7,-1        ;divide by 2, putting remainder bit in sign
        JUMPL 7,G0006   ;jump if linear subscript was odd
        HLRZ 3,@1(1)    ;fetch pointer from left half
        JRST G0007      ;jump to G0007
G0006:  HRRZ 3,@1(1)    ;fetch pointer from right half
G0007:  ...
```

If the compiler can determine at compile time that the linear subscript will always be odd or always even, it will simplify the code and omit the JUMPL, JRST, and the unused halfword fetch.

## SUMMARY

MacLISP supports the compilation of numerical programs into code comparable to that produced by a FORTRAN compiler while maintaining complete compatibility with the rest of the MacLISP system. All numeric code will run in the MacLISP interpreter; additional information may be given to the compiler in the form of declarations to help it generate the best possible code. If such declarations are omitted, the worst that happens is that the code runs slower.

Compatibility with non-numeric functions was achieved by the judicious choice of a uniform representation for LISP numbers combined with a compatible stack-allocated representation for temporary numeric values passed between functions. The use of stack allocation reduces the need for garbage collection of numbers, while the uniformity of representation eliminates the need for most run-time representation checks. One exception to this is that the use of stack-allocated numbers must be restricted; this difficulty is kept in check by maintaining a careful interface between safe and unsafe uses, and analyzing the safety of pointers as much as possible at compile time.

While numeric functions and non-numeric functions may call each other freely, a special interface is provided for one numeric function to call another in such a way as to avoid number consing.

Arrays are stored in such a way that they may be dynamically allocated and yet accessed quickly by compiled code. This is aided by the rich address arithmetic provided by the PDP-10.

The philosophy behind the implementation is that the generality of LISP and the speed of optimized numeric code are not incompatible. All that is needed is a well-chosen, uniform representation for data objects suitable for use by hardware instructions, combined with a willingness to handle important special cases cleverly in the compiler.

## REFERENCES

1. Fateman, Richard J.: "Reply to an Editorial." SIGSAM Bulletin 25 (March 1973), 9-11.

2. Teitelman, Warren: InterLISP Reference Manual. Revised edition. Xerox Palo Alto Research Center (Palo Alto, 1975).

# ON COMPUTING CLOSED FORMS FOR SUMMATIONS

Robert Moenck
Division of Physical Sciences
Scarborough College
University of Toronto

## ABSTRACT

The problem of finding closed forms for a summation involving polynomials and rational functions is considered. A method closely related to Hermite's method for integration of rational functions is derived. The method expresses the sum of a rational function as a rational function part and a transcendental part involving derivatives of the gamma function.

## Section 1. Introduction

Mathematicians have long been interested in finding closed form expressions for formal summations.

For example: $\sum_{i=1}^{n} i = \binom{n}{2}$

or

$$\sum_{i=1}^{n} \frac{i}{2^i} = 1 - \frac{n+1}{2^{n-1}} .$$

The history of this problem is dotted with the names of the giants of mathematics; names like Newton, Euler, Bernoulli or Boole. Jordan (ref. 1) gives a comprehensive survey of this field of mathematics. In spite of the many years of work which has been devoted to the problem, there is no general algorithmic approach to finding such closed forms. Jordan's book is more like a cookbook of approaches, rather than a rigorous algorithmic treatment, such as we would like to have for computer applications.

For this reason, since the turn of the century, the field has developed in other directions. In particular the areas of approximation theory and numerical analysis have been it's progeny. However, the need for finding closed forms for summations still exists. It is useful for large portions of the study of combinatorics. So, it would be nice, if the problem could be solved algorithmically, with the aid of algebraic manipulation. This paper is intended to lay some ground work to explore parts of the problem.

One reason that there is hope for an algorithmic solution, is the remarkable success in solving the integration problem. Work by mathematicians like Risch (ref. 2), Moses (ref. 3) and many others has resulted in the development of algorithms for finding closed forms for a large range of integrals. As Boole (ref. 4) noted in his work on differences over a century ago, there are strong parallels between the two problems. In this paper, we shall explore some of them and use the methods of the integration problem as a light to guide our way.

To a large extent the problem of finding closed forms for summations has been neglected in the work of algebraic manipulation. Johnson (ref. 5) considered the zero recognition problem for combinatorial sums and Gosper (ref. 6) considered the problem of automatically economizing summations. Recently, Cheatham (ref. 7) described a program which attempts to find a closed form for summations computed by loops in a program, and in reference 8 Gosper describes a method based on continued fractions, for finding summations.

In section (2) we present some notation and properties of differences. Section (3) sketches the summation of polynomials. Section (4) deals with finding the rational part of a summation of a rational function and section (5) briefly considers the transcendental part.

## Section 2. Some Notation

If we are presented with a definite summation and asked to find its closed form:

$$(1) \qquad g(n) = \sum_{i=a}^{n} f(i) \ ,$$

one way we can approach the problem is to find the indefinite summation:

$$h(x) = \sum_{i=0}^{x-1} f(i) \ .$$

Then one can evaluate h(x) to obtain g(n).

$$g(x) = \Big[ h(x) \Big]_{x=a+1}^{x=n+1} \ .$$

This brief sketch sidesteps the issue of any singularities which may occur in the function over the range of summation. However, it does point out the importance of the indefinite summation, the quantity we shall be concerned with here.

Implicit in our notation for (eq. 1) is that i takes on integral values between a and b. Therefore, if we take the first difference of h(x):

$$(2) \qquad \Delta h(x) = h(x+1) - h(x) = f(x)$$

we obtain f(x), the function we are trying to sum. Conversely, if we apply the inverse difference operator $\Delta^{-1}$ to f(x):

$$\Delta^{-1} f(x) = h(x)$$

we obtain the indefinite summation.

This leads to our first parallel between summation and integration: we can obtain an expression for the summation by anti-differencing the function; much in the way one obtains an integral by anti-differentiation. Also, the study of differences leads to the understanding of sums, much in the way differentiation is the key to integration.

The anti-difference is unique up to the addition of functions whose first difference is zero. Examples of such functions are:

a) constants

b) functions with period 1 e.g. $\sin(\pi x)$.

Since the beginning of the study of differences, it has been convenient to employ an operator notation to express equations. We shall use the notation employed by Jordan (ref. 1), which is fairly standard. The common and most useful operators are:

a) the Shift Operator $E : Ef(x) = f(x+1)$

b) the Difference Operator $\Delta : \Delta f(x) = Ef(x) - f(x)$

c) the inverse difference operator $\Delta^{-1} : \Delta^{-1} f(x) = \sum_{i=0}^{x-1} f(i)$

We will use the inverse difference operator $\Delta^{-1}$ to represent the quantity we wish to compute, to avoid any confusion between it and any bounded sums which will be expressed by the summation operator $\Sigma$. Occasionally, we shall extend the notation by indicating the variable involved in the difference and the length of the difference:

i.e.: $\underset{h}{\underset{x}{\Delta}} f(x,y) = f(x+h,y) - f(x,y)$.

Normally x will be understood from the context and h=1, and so this extra embellishment will not be necessary.

In modern terms operators a) and b) are derivations on an extension field $F(x,x_1,\ldots,x_n)$ over some ground field $F(x)$. Using these derivations Cohn (ref. 9) constructs a Difference Algebra much like Ritt's (ref. 10) Differential Algebra. However, Cohn is more concerned with the larger problem of systems of difference equations, rather than the simple linear difference equation (eq. 2).

## Properties of Differences:

The following properties can be simply derived from the definition of differences:

P1) $\Delta k f(x) = k \Delta f(x)$ , $k \in F$

P2) $$\Delta(f(x)+g(x)) = \Delta f(x)+\Delta g(x)$$

P3) $$\Delta(f(x)\cdot g(x)) = f(x)\cdot\Delta g(x) + Eg(x)\cdot\Delta f(x)$$

P4) $$\Delta\left(\frac{1}{g(x)}\right) = -\frac{\Delta g(x)}{g(x)\,Eg(x)}$$

P5) $$\Delta\left(\frac{f(x)}{g(x)}\right) = -\frac{f(x)\Delta g(x)+g(x)\Delta f(x)}{g(x)\,Eg(x)}$$

P6) $$\underset{1}{\overset{x}{\triangle}}(f(g(x))) = \underset{\Delta g(x)}{\overset{g(x)}{\triangle}}\ f(g(x))$$

It is the slight discrepancies between these properties and their analogous ones in differential algebra, that prevents direct application of its results and methods.

## Section 3. Sums of Polynomials

The simplest form of function we might want to sum is a polynomial:

$$a(x) = \Sigma\, a_i\, x^i\ .$$

In the case of differential algebra, the integral is easily obtained since:

(3) $$D\, x^n = n\, x^{n-1}\ .$$

Therefore, the integral is constructed by anti-differentiation. However, differences of powers do not have such a concise form:

$$\Delta x^n = \sum_{i=0}^{n-1} \binom{n}{i}\, x^{n-i}\ .$$

Thus expressing a function as a sum of powers is not a convenient form in difference algebra. Instead, the factorial functions are used:

(4) $$[x]_n = x(x-1)(x-2)\ \ldots\ (x-n+1)$$

The difference of a factorial is:

(5) $$\Delta[x]_n = E[x]_n \quad [illegible]_n = n[x]_{n-1}\ .$$

This has the concise form of (eq. 3) and so is a better representation. We can convert a polynomial to the factorial form using Newton's formula, which expresses a function in terms of it's higher differences:

(6) $$f(x) = \sum_{i=0}^{n} \frac{[x]_i}{i!}\, \Delta^i f(0)$$

where f(x) is a polynomial of degree n. The higher differences can be found using a difference table after evaluating the polynomial at the points x=0,1,...,n. Now:

$$(7) \qquad f(x) = \sum_{i=0}^{n} \frac{[x]_i}{i!} f_i$$

and so:

$$(8) \qquad \Delta^{-1}f(x) = \sum_{i=0}^{n} \frac{[x]_{i+1}}{(i+1)!} f_i = \sum_{i=0}^{n} \binom{x}{i+1} f_i$$

eg: To compute $g(x) = \Delta^{-1}(3x^3-2x+1) = \Delta^{-1}f(x)$

The difference table is:

| x | f(x) | Δf(x) | $\Delta^2 f(x)$ | $\Delta^3 f(x)$ |
|---|---|---|---|---|
| 0 | 1 | 1 | 18 | 18 |
| 1 | 2 | 19 | 36 | |
| 2 | 21 | 55 | | |
| 3 | 76 | | | |

and so

$$\begin{aligned}\Delta^{-1}f(x) &= \Delta^{-1}\left[\binom{x}{0} + \binom{x}{1} + 18\binom{x}{2} + 18\binom{x}{3}\right]\\ &= \binom{x}{1} + \binom{x}{2} + 18\binom{x}{3} + 18\binom{x}{4}\\ &= \Delta^{-1}(1+[x]_1+9[x]_2+3[x]_3)\\ &= \Delta^{-1}([x]_1+\tfrac{1}{2}[x]_2+3[x]_3+\tfrac{3}{4}[x]_4)\end{aligned}$$

To convert from factorial representation to power representation we can expand the factorial functions using their definitions.

$$[x]_1 = x$$

$$\tfrac{1}{2}[x]_2 = -\tfrac{1}{2}x + \tfrac{1}{2}x^2$$

$$3[x]_3 = 6x - 9x^2 + 3x^3$$

$$\tfrac{3}{4}[x]_4 = -\tfrac{9}{2}x - \tfrac{21}{4}x^2 - \tfrac{9}{2}x^3 + \tfrac{3}{4}x^4$$

$$\text{Total } g(x) = 2x - 13\tfrac{3}{4}x^2 - \tfrac{3}{2}x^3 + \tfrac{3}{4}x^4$$

## Section 4. Sums of Rational Functions

The next larger class of problems is sums of rational functions. In integration, these are approached using Hermite's method which performs a partial fraction decomposition of the function. Moses (ref. 3) describes this process. The partial fraction decomposition breaks the rational function into a sum of rational functions whose denominators are powers of square free factors of the original denominator. Then using integration-by-parts the integral can be expressed as a rational function portion and a transcendental portion which is a sum of logarithms.

We shall follow this method, with slight modifications, to derive a rational portion of the summation and a transcendental portion. The match of the two methods is close enough that we can describe it as Hermite Summation.

Remembering from section 3 that powers are not nice forms for summation, we define a factorial operator on a function:

$$[f(x)]_k = f(x)\cdot f(x-1)\cdot f(x-2)\ldots f(x-k+1) \quad \text{for } k>0 \ . \tag{9}$$

We can extend this operator by noticing:

$$[f(x)]_k = [f(x)]_\ell \cdot [f(x-\ell)]_{k-\ell} \tag{10}$$

If we define $[f(x)]_0 = 1$ and assert that (10) is an identity then substituting $k=0$ we get:

$$[f(x)]_{-\ell} = \frac{1}{[f(x+\ell)]_\ell} \tag{11}$$

We will call the value of $k$ or $\ell$ in equations 9 and 11, the factorial degree of function, because of its parallel to the "power" degree. We now proceed to examine the differences of factorials.

$$\Delta[f(x)]_k = [f(x)]_{k-1} \underset{k}{\triangle\!\!\!\!\times} f(x-k+1) \ , \ k>0 \ . \tag{12}$$

A special case of this is eq. 5 for factorial polynomials.

$$\begin{aligned} \Delta[f(x)]_{-\ell} &= -[f(x)]_{-(\ell+1)} \underset{\ell}{\triangle\!\!\!\!\times} Ef(x) \\ &= \frac{\underset{\ell}{\triangle\!\!\!\!\times} Ef(x)}{[f(x+\ell+1)]_{\ell+1}} \end{aligned} \tag{13}$$

Notice that the factorial degree is decreased (resp. increased) by 1 on differencing factorials (resp. reciprocal factorials).

Shift Free Decomposition:

If we are given a product of functions we can decompose it into a product of factorial functions. Suppose our product is of the form:

$$S = a \cdot b \cdot c$$

where a,b,c are mutually relatively prime and Ea=b. Then:

$$ES = (Ea)\cdot(Eb)\cdot(Ec) = b(Eb)\cdot(Ec)$$

and $$GCD(S,ES) = b$$

so we can divide out b and a from S and form:

$$S = [b]_2 \cdot C \ .$$

Applying this method repeatedly we can put a product into the form:

$$S = [S_1]_1 \cdot [S_2]_2 \ \dots \ [S_k]_k \ . \tag{14}$$

where the individual $S_i$ are shift-free. Given a rational function we can perform a shift-free partial fraction decomposition:

$$\frac{A(x)}{S(x)} = \sum_{i=1}^{k} \frac{A_i}{[S_i]_i} \tag{15}$$

and also a complete shift-free partial fraction decomposition.

$$\frac{A(x)}{S(x)} = \sum_{i=1}^{k} \sum_{j=1}^{i} \frac{A_{i,j}}{[S_i]_j} = \sum_{i=1}^{k} \sum_{j=1}^{i} \frac{A_{i,j}(x)}{[f_i(x+i)]_j} \ . \tag{16}$$

This complete shift-free partial fraction decomposition is completely analogous to the starting point of the integration-by-parts phase of Hermite's method. It can be computed in the same way the complete square free partial fraction decomposition for integration is done (see eg. Horowitz ref. 11 or Yun ref.12). We can also deduce $(f(x+k),\ \underset{k-1}{\Delta}\ f(x+1)) = 1$ iff $(f(x+k),\ f(x+1)) = 1$. This will be true if we have performed a k-shift-free decomposition of f(x).

Shift Independence:

We can test if a function is shift free using the GCD construction above. However this does not eliminate all the cases. Consider:

$$\frac{A(x)}{S(x)} = \frac{1}{x(x+3)}$$

Our GCD test will say S(x) is 1-shift-free which might lead to errors if we assume it is k-shift-free for all $k \epsilon Z$. We might call such function shift dependent since it is not 3-shift-free. We can test for shift independence using the following method:

1) Form S(x+k) where k is a new variable.
$$S(x+k) = x^2 + (2k+3)x + (k^2+3k) \ .$$

2) Compute the resultant with respect to k:

$$\text{Res}\,(S(x+k),\ S(x)) = R(k)$$
$$\text{Res}\,(x^2+(2k+3)x+(k^2+3k),\ x^2+3x) = -k^4-9k^2$$

3) Test for integer roots of R(k); these will disclose any k's with non-trivial GCD's of the form.

$$\text{GCD}\,(S(x),\ E^k S(x)).$$

i.e.: $k=0,\pm 3$. Choose: $k=+3$.

4) Apply Stirling's Method to convert the rational function into a factorial denominator. i.e. multiply top and bottom by $(x+1)(x+2)$ to obtain

$$\frac{A(x)}{S(x)} = \frac{(x+1)(x+2)}{[x+3]_4}\ .$$

5) Proceed as before.

Summation by Parts

From property P3) of differences we can deduce the rule for summation-by-parts:

$$\Delta^{-1}\,(u\cdot\Delta v) = u\cdot v - \Delta^{-1}[Ev\ \Delta v] \tag{17}$$

We can apply this to a typical term in our complete shift-free partial fraction decomposition.

$$\Delta^{-1}\,\frac{A_{i,j}(x)}{[f_i(x+i)]_j}\ ,\quad j\le i$$

First we can apply the extended euclidean algorithm to find B,C such that:

$$B\ f_i(x+i-j+1) + C\ \underset{j-1}{\triangle\!\!\!\!\!\times}\ f(x+i-j) = 1. \tag{18}$$

This can be used to expand the term further as:

$$\begin{aligned}\Delta^{-1}\,\frac{A_{i,j}(x)}{[f_i(x+i)]_j} &= \Delta^{-1}\frac{A_{i,j}\ C\,\underset{j-1}{\Delta}f(x+i-j)}{[f_i(x+i)]_j} + \Delta^{-1}\frac{A_{i,j}\ Bf(x+i-j+1)}{[f_i(x+i)]_j} \\ &= \Delta^{-1}\frac{D\,\underset{j-1}{\Delta}f(x+i-j)}{[f_i(x+i)]_j} + \Delta^{-1}\frac{A_{i,j}\ B}{[f_i(x+i)]_{j-1}}\ .\end{aligned} \tag{19}$$

Applying summation by parts to the first term of eq. 19,

$$\begin{aligned}\Delta^{-1}\left(\frac{D\cdot\underset{j-1}{\Delta}f(x+i-j)}{[f_i(x+i)]_j}\right) &= \frac{-D}{[f_i(x+i-1)]_{j-1}} - \Delta^{-1}\left(\frac{-\Delta D}{E[f(x+i-1)]_{j-1}}\right) \\ &= \frac{-D}{[f_i(x+i-1)]_{j-1}} + \Delta^{-1}\left(\frac{\Delta D}{[f(x+i)]_{j-1}}\right)\end{aligned} \tag{20}$$

The second terms of (20) and (19) and any terms of factorial degree j-1 in the complete shift-free partial fraction decomposition, can be combined together to give the next term of the iteration:

$$\Delta^{-1}\left(\frac{\Delta D+A_{i,j}B+A_{i,j-1}}{[f_i(x+i)]_{j-1}}\right) .$$

The same method can be applied again. Continuing in this way we eventually obtain an expression for the indefinite sum of a rational function as a rational function plus an indefinite summation of terms with shift-free denominators of factorial degree 1.

An Example of Hermite Summation:

We wish to compute: $\Delta^{-1}\frac{A}{B}$

where:

$$\frac{A}{B} = \frac{-(x^2+3x+3)}{x^4+2x^3-3x^2-4x+2} .$$

First we put B into a shift free form:

$$EB = x^4+6x^3+9x^2-2$$

and

$$GCD(B,EB) = (x^2+2x-1)$$

and so $\frac{A}{B} = \frac{-(x^2+3x+3)}{[x^2+2x-1]_2}$ .

Next we perform a complete shift-free decomposition on $\frac{A}{B}$ :

$$\frac{A}{B} = \frac{-(3x+5)}{[x^2+2x-1]_2} + \frac{-1}{[x^2+2x-1]_1} = \frac{C}{D} + F .$$

Now we want to put $\frac{C}{D}$ into a form suitable for summation by parts. Since $E^{-1}(x^2-2x-1) = x^2-2$.

$$\frac{C}{D} = \frac{G\Delta(x^2-2)}{[x^2+2x-1]_2} + \frac{H(x^2-2)}{[x^2+2x-1]_2}$$

Since $(x^2-2)$ is shift free:

$$(\Delta(x^2-2),\ (x^2-2)) = 1$$

and therefore we can employ the extended euclidean algorithm to solve the equation:

$$-(3x+5) = S(2x+1) + T(x^2-2)$$

$$= -(x+1)(2x+1) + 2(x^2-2)$$

So $\frac{C}{D}$ is of the form:

$$\frac{C}{D} = \frac{-(x+1)(2x+1)}{[x^2+2x-1]_2} + \frac{2(x^2-2)}{[x^2+2x-1]_2}$$

$$\frac{-(x+1)\Delta(x-2)}{[x^2+2x-1]_2} + \frac{2}{[x^2+2x-1]_2} .$$

Now we perform summation by parts to obtain:

$$\Delta^{-1}\frac{C}{D} = \frac{x+1}{(x^2-2)} - \Delta^{-1}\left\{\frac{\Delta(x+1)}{E(x^2-2)}\right\} + \Delta^{-1}\frac{2}{[x^2+2x-1]_1}$$

and so:

$$\Delta^{-1}\frac{A}{B} = \Delta^{-1}\frac{C}{D} + \Delta^{-1}F = \frac{x+1}{x^2-2} + \Delta^{-1}\left\{\frac{-1}{[x^2+2x-1]_1} + \frac{2}{[x^2+2x-1]_1}\right.$$

$$\left. - \frac{1}{[x^2+2x-1]_1}\right\} = \frac{x+1}{x^2-2} .$$

## Section 5: The Transcendental Part

We have reduced the problem of summation of rational functions to the summation of a set of terms with shift-free denominators. Now we define a set of functions:

$$\psi_m(x) = D^m \log\Gamma(x+1) \text{ , } m>0$$

where $\Gamma(x)$ is the gamma-function, a generalization of the factorial. The functions $\psi_m$ have the property:

$$\begin{aligned}\Delta\psi_m(x) &= D^m \Delta\log\Gamma(x+1)\\ &= D^m \log\frac{\Gamma(x+2)}{\Gamma(x+1)} = D^m \log(x+1)\\ &= D^{m-1}\frac{1}{x+1} = \frac{(-1)^{m-1}(m-1)!}{(x+1)^m} .\end{aligned}$$

Therefore the sum of a negative power of (x+1) is:

$$\Delta^{-1}\frac{1}{(x+1)^m} = \frac{(-1)^{m-1}}{(m-1)!}\psi_m(x) .$$

The functions $\psi_m(x)$ are also known as the polygamma functions.

We can now expand the remainder of our rational function in terms of its roots:

$$\frac{A(x)}{B(x)} = \sum_{i=1}^{k} \frac{a_i}{(x-b_i)^{j(i)}}$$

where $j(i)$ is the multiplicity of the root.

Using the $\psi_m$ functions the indefinite summation of remainder of the rational function is:

$$\Delta^{-1} \frac{A(x)}{B(x)} = \sum_{i=1}^{k} \frac{a_i(-1)^{j(i)-1}}{(j(i)-1)!} \psi_{j(i)} (x-b_i-1)$$

The functions $\psi_m$ play a role similar to logarithms in the integration of rational functions. I conjecture:

a) The functions $\psi_m(x)$ are transcendental with respect to the ground field $F(x)$.

b) If $b_i$ are the shift-free roots of a polynomial then $\psi_{j(i)}(x-b_i)$ are algebraically independent.

If these statements are true then one could argue, much as Hermite did for integration, that the rational and transcendental parts of a summation are unique.

# REFERENCES

1. Jordan, C.: Calculus of Finite Differences. Chelsea, N.Y., 1965, 3rd ed.

2. Risch, R.: The Problem of Integration in Finite Terms. Trans. A.M.S.139, May 1969, pp. 162-189.

3. Moses, J.: Symbolic Integration: the Stormy Decade. Comm. A.C.M., vol. 14 no. 8, Aug. 1971, pp. 548-560.

4. Boole, G.: A Treatise on the Calculus of Finite Differences. Cambridge, 1860.

5. Johnson, S.C.: On the Problem of Recognizing Zero. J.A.C.M., vol. 18, no. 4, Oct. 1971, pp. 559-565.

6. Gosper, R.W.: A Calculus of Series Rearrangements. Proc. Symp. on New Directions in Algorithms and Complexity, C.M.U. 1976.

7. Cheatham, T.E. and Townley, J.A.: Symbolic Evaluation of Programs: A Look at Loop Analysis. Proc. 1976 Symp. on Symbolic and Algebraic Computation, R.D. Jenks Ed., A.C.M. New York, pp. 90-96.

8. Gosper, R. Wm., Jr.: Indefinite Hypergeometric Sums in MACSYMA. Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper no. 24 of this compilation.)

9. Cohn, R.M.: Difference Algebra. Interscience, N.Y., 1965.

10. Ritt, J.F.: Differential Algebra. Coll. A.M.S., vol. 33, 1950.

11. Horowitz, E.: Algorithms for Partial Fraction Integration. Proc. 2nd Symp. on Symbolic and Algebraic Manipulation, Max 1971. S.R. Petrick ed., pp. 441-457.

12. Yun, D.: On Partial Fraction Decomposition Algorithms. Proc. 1977 IFIP Congress.

# Indefinite Hypergeometric Sums in MACSYMA[*]

R. Wm. Gosper, Jr.
XEROX Palo Alto Research Center

## ABSTRACT

We present a MACSYMA function which, given the summand

(A) $$a_n = \Delta g(n) = g(n+1) - g(n)\,,$$

finds $g(n)$, the "indefinite sum", within an additive constant, provided that $g(n+1)/g(n)$ is a rational function of $n$. We then have the identity

(B) $$\sum_{n=p}^{q} a_n = g(q+1) - g(p)\,.$$

Examples:

$$\sum_{n=0}^{m} \frac{1}{n^2+\sqrt{5}n-1} = \frac{\sqrt{5}-3}{6} - \frac{2}{3}\left(\frac{1}{2m+\sqrt{5}-1} + \frac{1}{2m+\sqrt{5}+1} + \frac{1}{2m+\sqrt{5}+3}\right),$$

$$\sum_{n=0}^{m} \frac{n(n+a+b)a^n b^n}{(n+a)!\,(n+b)!} = \frac{1}{(a-1)!\,(b-1)!} - \frac{a^{m+1}b^{m+1}}{(m+a)!\,(m+b)!}\,,$$

$$\sum_{n=0}^{m} \frac{n^4 4^n}{\binom{2n}{n}} = \frac{2}{693}\left(\frac{(m+1)(63m^4+112m^3+18m^2-22m+3)4^m}{\binom{2m}{m}} - 3\right),$$

$$\sum_{n=0}^{m} \frac{(3n)!}{n!\,(n+1)!\,(n+2)!\,27^n} = \frac{(81m^2+261m+200)\,(3m+2)!}{40m!\,(m+1)!\,(m+2)!\,27^m} - \frac{9}{2}\,.$$

The algorithm seeks a "telescoping function" $f(n)$ satisfying

(C) $$g(n) = f(n)\,a_n\,,$$

which, if used to eliminate $g$ from (A), yields the functional equation

(func) $$\frac{a_{n+1}}{a_n} = \frac{f(n)+1}{f(n+1)}\,,$$

whence, from (B) and (C),

(voila) $$\sum_{n=p}^{q} a_n = a_q(f(q)+1) - a_p f(p)\,.$$

From (A) and (C) it can be shown that $f(n)$ is a rational function if $g(n+1)/g(n)$ is. Our algorithm determines $f$ as a finite continued fraction whose terms are polynomials in $n$. We await either a mathematical proof of its effectiveness, or alternatively, an example on which it fails, to determine whether it is a decision procedure, or merely a useful but fallible heuristic.

---

[*] This work was supported, in part, by the National Science Foundation, and was fostered by the hospitable and unfettered environment at the Stanford Artificial Intelligence Laboratory.

If $g(n+1)/g(n)$ is a rational function of $n$, then $g(n)$, and therefore $a_n = \Delta g(n)$, is a constant times a product of $n$ consecutive values of some rational function. We shall call such functions "hypergeometric terms". We believe our algorithm finds all inverse differences which have this form, thus performing indefinite summation on generalized hypergeometric series.

Of course, not all finite products of rational functions sum to functions of the same type, just as not all rational functions integrate to rational functions. One might ask, therefore, whether precluding higher functions from the answer $g(n)$ might thwart our algorithm the way precluding logarithms, etc., would thwart an integration algorithm. The answer is yes, but not nearly as badly. It appears that among the familiar higher functions, only the polygammas* of certain linear arguments have first differences in the form of hypergeometric terms. This paucity of functions applicable to the expression of indefinite sums is due to the lack of a discrete analogue to the chain rule, and has the unfortunate consequence that a given sum is less likely to have a closed form than is an integral of similar complexity. In particular, the only hypergeometric series whose indefinite sums are facilitated by polygammas are those with rational summands. Should it be needed, a fairly simple partial fractions algorithm can sum rational functions as polygammas, at least when it is clear how to adequately factor the summand's denominator. (Polygammas in the summands might be handled using summation by parts, but not in the algorithm under discussion.)

It is a little surprising that the rational summands which require polygammas are invariably special cases of hypergeometric summands which are amenable to our MACSYMA sum function, e.g.

$$\sum_{n=1}^{m} \frac{1}{n^2} = \psi'(1) - \psi'(m+1)$$

will simplify no further ($\psi'$ is the trigamma function), yet this sum is the special case $x \to 0$ of

$$\sum_{n=1}^{m} \frac{(n-1)!^2}{(n-x)!\,(n+x)!} = \frac{m!^2}{x^2(m-x)!\,(m+x)!} - \frac{\sin \pi x}{\pi x^3} .$$

Here the telescoping function was $f(n) = n^2/x^2 - 1$. (We also used the factorial reflection formula, $x!\,(-x)! = \pi x/\sin \pi x$.) In general, the telescoping function $f(n) = -1/\epsilon a_n$ yields the identity

$$\sum_{n=0}^{m} a_n(1-\epsilon a_0)(1-\epsilon a_1)\cdots(1-\epsilon a_{n-1}) = \frac{1-\prod_{n=0}^{m}(1-\epsilon a_n)}{\epsilon} .$$

Letting $\epsilon \to 0$, we have an arbitrary sum as the limit of a product over the same range (which is clear from considering the expansion of the product through the $O(\epsilon)$ terms.) When $a_n$ is rational in $n$, we can always express this product and the summand as hypergeometric terms prior to taking the limit. Thus, for another example of the sum of reciprocal squares, use $a_n = 1/n^2$ (and, for convenience, replace $\epsilon$ by $\epsilon^2$):

$$\sum_{n=0}^{m-1} \frac{1}{(n+1)^2\binom{n}{\epsilon}\binom{n}{-\epsilon}} = \frac{1}{\epsilon^2} - \frac{(m-\epsilon)!\,(m+\epsilon)!\,\sin \pi\epsilon}{m!^2\pi\epsilon^3} .$$

(The value $\zeta(2) = \pi^2/6$ is evident if $m \to \infty$ before $\epsilon \to 0$.)

---

*derivatives of log $\Gamma(x)$

Unfortunately, the current algorithm is not a decision procedure for the expressibility of indefinite hypergeometric sums in closed form. The top level procedure heuristically bounds the complexity of the telescoping function $f$, to prevent the main iteration, when given an impossible problem, from plunging down an endless continued fraction. Another as yet nonrigorous aspect of the main iteration: it uses a rather shortsighted, "greedy" algorithm to determine the successive term polynomials, and we have yet to show that it will never need to backtrack when solving the functional equations which arise from series. (If necessary, backtrack could be installed, but it might be very costly in cases which turn out inexpressible in closed form.)

## The Algorithm

The only significant problem is to solve the rational functional equation

(func) $$\frac{a_{n+1}}{a_n} = \frac{f(n)+1}{f(n+1)}$$

for $f$. Since this followed from differencing $g(n) = f(n)\,a_n = f(n)\Delta g(n)$, we have

$$f(n) = \frac{g(n)}{g(n+1)-g(n)} = \frac{1}{\frac{g(n+1)}{g(n)} - 1}$$

which is rational when $g(n+1)/g(n)$ is. Because we have no boundary condition to satisfy, equation (func) is easier to satisfy than a first order linear recurrence with polynomial coefficients. In fact, if $f(n)$ is a solution, then so is $f(n)+c/a_n$, $c$ arbitrary. Thus if the summand $a_n$ is rational, then there is a continuum of rational $f$ satisfying (func), differing only in the "constant of summation" $c$ that they add to the sum $g$.

If $f$ is a rational function, then the quotients from Euclid's algorithm (using polynomial division) form the terms of its continued fraction:

$$f(n) = p_1(n) + \cfrac{1}{p_2(n) + \cfrac{1}{p_3(n) + \cfrac{1}{\ddots \; p_k(n)}}} .$$

Our MACSYMA algorithm successively determines $p_1, p_2, \ldots$, with the proviso that no $p_i$ be constant for $i >$ 1, so as to guarantee the uniqueness of the representation.

Since the term ratio $a_{n+1}/a_n$ is a rational function, we can write it as $P(n)/Q(n)$, where $P$ and $Q$ are polynomials. Then $f$ must satisfy

(1) $$P(n)f(n+1) - Q(n)(f(n)+1) = 0 .$$

In particular, this relation holds for large $n$, where $f(n) \to p_1(n)$. We thus "greedily" determine $p_1$ as the polynomial approximation to $f$ which most nearly satisfies (1), that is, the polynomial which minimizes the degree of the lefthand side. We then substitute $p_1(n) + 1/f_2(n)$ for $f(n)$, so that we can recursively determine the rest of $f$'s continued fraction as $f_2$, the solution of the new functional equation

$$\frac{a_{n+1}}{a_n} = \frac{p_1(n) + \dfrac{1}{f_2(n)} + 1}{p_1(n+1) + \dfrac{1}{f_2(n+1)}} .$$

We write this equation in the form

(2ndform) $$A(n)f_2(n)f_2(n+1) + B(n)f_2(n) + C(n)f_2(n+1) + D(n) = 0 ,$$

where $A$, $B$, $C$, and $D$ are polynomials. Then we "greedily" seek the polynomial $p_2$ which, in place of $f_2$, most nearly satisfies (2ndform). We proceed in this way, replacing

(subst) $$f_i(n) \text{ by } p_i(n) + 1/f_{i+1}(n)$$

until we either find a $p_k(n)$ which exactly satisfies our equation, or we conclude that no solution exists. Fortunately, further substitutions of the form (subst) lead to no equations more complicated than (2ndform).

Worked Example: we seek

$$\sum_{n=0}^{m} (n-r/2)\binom{r}{n}$$

in closed form. Equation (func) becomes

$$\frac{n-r/2+1}{n-r/2}\,\frac{r-n}{n+1} = \frac{f(n)+1}{f(n+1)}$$

or

(f1) $$(n-r/2+1)(n-r)f(n+1) + (n-r/2)(n+1)(f(n)+1) = 0 .$$

In order to determine the first polynomial of $f$'s continued fraction, we must first determine the degree of that polynomial. We do this by replacing $f$ with the "polynomial" estimate $p_1(n) = an^q+O(n^{q-1})$, $q$ to be determined. Suppose $q>0$. Then (f1) becomes

$$2an^{q+2} + O(n^{q+1}) = 0 ,$$

implying $a = 0$, meaning that $q$ was too large. So $q$ must be 0, and thus $p_1$ must be a constant $a$, making (f1)

$$(2a+1)n^2 + O(n) = 0 ,$$

which determines $a = -1/2$ and therefore $f(n) = -1/2 + 1/f_2(n)$. This determines the new equation

(f2) $$r(n-\frac{r+1}{2})f_2(n+1)f_2(n) + 2(n-\frac{r}{2})(n+1)f_2(n+1) + 2(n-r)(n-\frac{r}{2}+1)f_2(n) = 0$$

to be solved for $f_2$. Again, if we estimate $f_2$ by $p_2(n) = an^q + O(n^{q-1})$, (f2) becomes

$$ra^2n^{2q+1} + 4an^{q+2} + O(n^{2q}) + O(n^{q+1}) = 0 .$$

Now $q$ must be positive since we have forbidden $p_i$ to be constant for $i>1$. But if $q>1$ then $2q+1 > q+2$, forcing $a$ to be 0, which is equivalent to reducing $q$. So $q=1$, and the above becomes

$$(ra^2+4a)n^3 + O(n^2) = 0$$

which determines $a = -4/r$. Now we know that $p_2(n)$ is of the form $-4n/r + b$, and we can determine $b$ by substituting this expression for $f_2$ in (f2), leaving for the lefthand side

$$(2-b)\left(4n^2 + (4-br-2r)n + b\frac{r(r-1)}{2} + 2\right),$$

which identically vanishes if $b = 2$. Thus we have found the solution

$$f(n) = -\frac{1}{2} + \frac{1}{-\frac{4n}{r} + 2} = \frac{n}{r-2n}$$

whence, by

(voila) $$\sum_{n=p}^{q} a_n = a_q(f(q)+1) - a_pf(p) ,$$

we get

$$\sum_{n=0}^{m} (n-r/2)\binom{r}{n} = \frac{(m-r)\binom{r}{m}}{2} .$$

(This example was suggested by D. Knuth.) Incidentally, Euler, (refs. 1, 1a), had the special case $m = \infty$ of (voila) in 1753, but he didn't get much mileage out of it. Chrystal (ref. 2) gives (voila) within a change of variables, but still underestimates its generality. He credits Euler in "*Nov. Comm. Petrop.*, 1760", but I can't locate this.

In certain cases where the continued fraction fails to terminate quickly, it is possible to deduce the general formula for the $i$th term. With this you can tell if and when the fraction will terminate, and in any case get an interesting identity. Consider, for example,

(2F1) $${}_2F_1\left[{a,\ 1 \atop c};\ x\right] = \frac{\sum\limits_{n\ge0}\binom{n+a-1}{a-c}x^n}{\binom{a-1}{a-c}},$$

which encompasses the Taylor series of many useful functions, e.g.

$$(1-x)^p \qquad [a=-p,\ c=1,\ x=x]$$

$$-\frac{\ln(1-x)}{x} \qquad [a=1,\ c=2,\ x=x]$$

$$\frac{\arctan x}{x} \qquad [a=1/2,\ c=3/2,\ x=-x^2]$$

$$\frac{\arcsin x}{x\sqrt{1-x^2}} \qquad [a=1,\ c=3/2,\ x=x^2]$$

First off, we note that equation (voila) has arbitrary upper and lower limits on the sum. We exploit this degree of freedom by shifting the summation index by $c$-1, so that (2F1) becomes

$$ {}_2F_1\left[{a,\,1 \atop c};z\right] = \frac{z^{1-c}\sum\limits_{n\ge c-1}\binom{n+a-c}{a-c}z^n}{\binom{a-1}{a-c}}, $$

which, if we replace $a-c$ by $b$, eliminates a parameter from the summand. (Summing for $n\ge c-1$ means for $n$ = $c-1$, $c$, $c+1$, ... regardless of whether $c$ is integral or even real.)

Equation (func) now becomes

$$ \frac{n+b+1}{n+1}z = \frac{f(n)+1}{f(n+1)}. $$

Experience indicates that, having determined $p_i$ in the form $(An + B)/C$, say, we should clear out the denominator C by writing

$$ f_i(n) = \frac{1}{C}\Bigl(An + B + \frac{1}{f_{i+1}(n)}\Bigr) \quad \text{instead of} \quad \frac{An+B}{C} + \frac{1}{f_{i+1}(n)} $$

before going on to determine $f_{i+1}$. This will usually lead to simpler coefficients in the later polynomials. Our solution will then begin

$$ f(n) = \frac{1}{z-1}\left(1 + \cfrac{bz}{(1-z)n - bz + 1 + \cfrac{(b-1)z}{(1-z)n - (b-1)z + 2 + \cfrac{2(b-2)z}{(1-z)n - (b-2)z + 3 + \cfrac{3(b-3)z}{\ddots}}}}\right) $$

and, in general, the $i$th equation

$$ (-1)^i\Bigl((i-2)(b-i+2)zf_i(n)f_i(n+1) - (n+i-1)f_i(n+1) + (n+b-i+3)zf_i(n) - 1\Bigr) = 0 $$

determines

$$ f_i(n) = \frac{1}{(i-2)(b-i+2)z}\Bigl((1-z)n - (b-i+2)z + i - 1 + \frac{1}{f_{i+1}(n)}\Bigr) $$

which in turn yields the $i$+1st equation

$$ (-1)^{i+1}\Bigl((i-1)(b-i+1)zf_{i+1}(n)f_{i+1}(n+1) - (n+i)f_{i+1}(n+1) + (n+b-i+2)zf_{i+1}(n) - 1\Bigr) = 0, $$

for $i\ge 3$. Finally, since $a_0 = 1$ and (if the series converges) $a_n \to 0$,

$$ {}_2F_1\left[{a,\,1 \atop c};z\right] = \frac{1}{1-z}\left(1 + \cfrac{(a-c)z}{(1-a)z + c + \cfrac{(a-c-1)z}{(2-a)z + c + 1 + \cfrac{2(a-c-2)z}{(3-a)z + c + 2 + \cfrac{3(a-c-3)z}{\ddots}}}}\right). $$

By the same method we can also establish

$$\sum_{n=m}^{\infty} \frac{z^n}{n!} = \frac{z^m}{m!}\left(1 + \cfrac{z}{m + 1 - z + \cfrac{z}{m + 2 - z + \cfrac{2z}{m + 3 - z + \cfrac{3z}{\ddots}}}}\right),$$

which, for $m = 0$, gives a nice continued fraction for $e^z$.

## Messy Details

I have glossed over three problems that arise in determining the successive polynomials, namely, what degree polynomial to choose, how many coefficients must be solved for at once, and what to do about multiple solutions.

1) The polynomial degree:

The MACSYMA algorithm basically chooses the largest integer $q$ such that when $an^q$ is substituted for $f_i(n)$ in the expression

$$\text{(ithform)} \qquad A(n)f_i(n)f_i(n+1) + \frac{B(n)+C(n)}{2}(f_i(n+1)+f_i(n)) + \frac{B(n)-C(n)}{2}(f_i(n+1)-f_i(n)) + D(n) = 0\,,$$

more than one of the four lefthand terms is of maximal degree in $n$. When there is such a largest $q$, the coefficient of the highest power of $n$ will contain at least two different powers of $a$, so that the coefficient can be eliminated with a nonzero choice of $a$. But on the first term ($i = 1$), $A(n) = 0$ and $C(n) = D(n)$, and it can happen that deg $(B(n)+C(n))$ < deg $(B(n)-C(n))$, *i.e.* $B(n)$ and $C(n)$ have the forms

$$cn^p + d_B\, n^{p-1} + O(n^{p-2}) \quad \text{and} \quad -cn^p + d_C\, n^{p-1} + O(n^{p-2})$$

respectively. Then, since deg $(f_i(n+1)+f_i(n))$ = deg $(B(n)-C(n))$ + 1, there is no largest $q$ meeting our condition. In this case, we estimate $f_1(n) = an^q + bn^{q-1} + O(n^{q-2})$, and the quantity we are trying to annihilate becomes

$$\text{(tricky)} \qquad B(n)f_i(n+1) + C(n)f_i(b) + D(n) = a(cq+d_B+d_C)n^{p+q-1} - cn^p + O(n^{p+q-2}) + O(n^{p-1})\,.$$

Here we can zero the high order coefficient with either of two choices: $q = -(d_B+d_C)/c$ or $q = 1$. The program heuristically chooses the larger of these, provided it is an integer, on the theory that there is a good chance of later determining that $a = 0$, should the choice prove wrong. But this reasoning is questionable in light of the functional equation

$$\text{(loser)} \qquad ((f(n)-n^{105})^2 - n)\,(nf(n+1) - 1) = 0\,.$$

The two solutions of this equation are evidently

$$f(n) = n^{105} + \sqrt{n} \quad \text{and} \quad f(n) = \frac{1}{n-1}\,,$$

but only the second solution is a rational function. Thus, any attempt to find the first solution will result in nontermination. Yet the erroneous choice of $p_1(n) = n^{105}$ reduces the lefthand side of (loser) to degree 107, while the correct choice $p_1(n) = 0$ only reduces it to degree 210. Our meek excuse is that in problems arising from sums we never encounter such products as $(f(n+1))^2 f(n)$, which appears in (loser). (Robert Maas helped construct this example).

At the end of the next section, we give an example where the heuristic succeeds in retroactively determining that the high coefficient is 0, but very nearly requires backtracking to do it.

2) The need to consider more than one coefficient at a time:

The aforementioned (tricky) case, in which the exponent becomes involved in the coefficients, is the source of another, less serious annoyance. In this case, and this alone, it is necessary to determine each coefficient of $p_1$ in terms of the next lower one. Consider the sum

$$\sum_{n=0}^{m} \frac{\binom{2n}{n}^2}{(n+1)\,4^{2n}}$$

which determines the functional equation

$$\frac{(n+1/2)^2}{(n+1)(n+2)} = \frac{f(n)+1}{f(n+1)} .$$

In the notation of the preceding discussion, $c = 1$, $d_B = 1$, $d_C = -3$, and thus $q = 2$. Now suppose we estimated $f(n)$ as $an^2 + bn + O(1)$. Then we would have

$$\left(\frac{5a}{4} - b - 1\right)n^2 + O(n) = 0 .$$

Had we merely estimated $f(n)$ by $an^2$, we would have erroneously determined $a$ on the assumption that $b$ was 0, and then gone on to determine that $b$ was, in fact, nonzero. Since $a$'s value depends on $b$'s, this incorrect value of $a$ would fail to annihilate the $n^2$ term, leaving that job for $b$. If $c$ is expended on the linear term, it happens that the constant term remains unvanquished, and the continued fraction process will plunge down an almost certainly bottomless hole. This would be a shame, since the equation could have been solved with the first term:

$$f(n) = 4n^2 + 4n .$$

This, incidentally, provides

$$\sum_{n=0}^{m} \frac{\binom{2n}{n}^2}{(n+1)\,4^{2n}} = \frac{(m+1)\binom{2m+1}{m}^2}{4^{2m}} .$$

In principle, it is never necessary to solve simultaneous equations, even in this worst case. It is merely necessary to determine each coefficient in terms of the as yet undetermined succeeding coefficient, and only in those cases where $B(n)+C(n)$ has lower degree than $B(n)-C(n)$, and only for the first polynomial. In practice, our algorithm invokes MACSYMA's LINSOLVE linear system package, mainly for its automatic back substitution.

**Incidentally, the only way that a coefficient could depend on the next *two* coefficients would be if the functional equation contained three distinct invocations of $f$, say $f(n-1)$, $f(n)$, and $f(n+1)$.**

**Very occasionally, an equation for a coefficient can have no solutions! This happens while summing**

$$\sum_{n=0}^{m} \frac{(4n-1)\binom{2n}{n}^2}{(2n-1)^2 4^{2n}} \tag{weirdo}$$

**which requires the solution of**

$$\frac{(n-\frac{1}{2})^2(4n+3)}{(n+1)^2(4n-1)} = \frac{f(n)+1}{f(n+1)} .$$

**Proceeding as before, we would again find $q = 2$ and estimating $f$ by $an^2 + bn + O(1)$, we would determine that $a = -2-2b$. Then estimating $f = -2(b+1)n^2 + bn + c + O(n^{-1})$, we would determine $b = (16c+1)/3$. But this leaves the equation**

$$3n + 9c + 3 + O(n) = 0$$

**and there is no way to choose $c$ to annihilate the coefficient of $n$, since it depends on the next continued fraction term rather than on $c$. It is unsafe to choose $c$ arbitrarily, since our nonrational summand precludes the "constant of summation", so we must postpone the determination until after determining that the second continued fraction term is $(-16n+36c+13)/3$, which leaves us the lefthand side**

$$-3\,(4c+1)\,((180c+45)n - 324c^2 - 117c + 16) .$$

**Our patience is rewarded, for the determination $c = -\frac{1}{4}$ terminates the problem, but with the ironic result that $b = \frac{4}{3}$ and $a = 0$, so that the choice $q = 1$, which is always available in such cases, was correct after all. (See the first sentence after equation (tricky).) Incidentally, we have determined**

$$f(n) = -\frac{4n^2}{4n-1}$$

**and thus**

$$\sum_{n=0}^{m} \frac{(4n-1)\binom{2n}{n}^2}{(2n-1)^2\, 4^{2n}} = -\frac{\binom{2m}{m}^2}{4^{2m}} .$$

**3) Multiple roots when determining a coefficient:**

**If $f(n)$ is a rational function with rational coefficients, we can be sure that no irrational coefficient will arise in its continued fraction. It is therefore reasonable to hope that in solving a functional equation for such a continued fraction, no nonlinear equation need be solved. This hope is substantially fulfilled, but for a couple of glitches. For example, in establishing the identity**

$$\sum_{n=0}^{m} n^4 = \frac{m(m+1)(2m+1)(3m^2+3m-1)}{30} ,$$

we would determine the telescoping function to be

$$f(n) = \frac{n}{5} - \frac{1}{2} + \cfrac{1}{3n + \cfrac{3}{10n + c + \cfrac{(c^2-10)^2}{10(c^2-10)n - c^3 + \cfrac{c^4}{(c^2-10)n + c}}}} = \frac{(n-1)(2n-1)(3n^2-3n-1)}{30n^3} + \frac{c}{300n^4}$$

where $c$ is the arbitrary "constant of summation" which we get when the summand is rational. But our algorithm is not smart enough to leave $c$, (which is also the coefficient of $n^0$ in $p_3(n)$), undetermined, and the consequences of this greed can be annoying. To determine the linear coefficient in $p_3$, $an$ is substituted for $f_3(n)$ in the current (i.e. the third) equation, resulting in a polynomial of the form $a(10-a)n^4 + O(n^3)$, which determines $a = 10$. But then, when we go to determine $c$ by estimating $f_3(n)$ as $10n + c$, we find we have a polynomial of the form $(c^2-10)n^2 + O(n)$. In other words, the choice $a = 10$ "fortuitously" annihilated the cubic, as well as the quartic term. Ordinarily, the only quadratic equations we encounter are of the degenerate form $a(k-a) = 0$, which occur when we are determining the high coefficient of each $p_i$ after $i = 1$. If choosing $a$ (or any lower coefficient) annihilates only one term of the expression being reduced, then the next term cannot be quadratic in the coefficient below $a$. This is because squares of coefficients of $f$ can only come from the $f(n)f(n+1)$ term of the functional equation, but here the first quadratic instances of each coefficient come two powers of $n$ apart. But when two or more powers of $n$ disappear with one choice of coefficient, we may be left with a nondegenerate quadratic equation for the next coefficient.

Greedily pursuing our example, then, we find $c = \sqrt{10}$, which makes our continued fraction for $f$ an indeterminate form. Either by performing the algorithm or taking limits, we find that the continued fraction found by the greedy algorithm is actually one term shorter:

$$f(n) = \frac{n}{5} - \frac{1}{2} + \cfrac{1}{3n + \cfrac{3}{10n + \sqrt{10} + \cfrac{1}{\sqrt{10}\,n^2 - n}}}.$$

Although MACSYMA solves quadratic equations as readily as linear ones, the introduction of surds into the computation can consume valuable time and storage, especially if it happens more than once, or involves large expressions containing symbolic parameters. If the original sum was rational and involved no surds, yet a surd arises in the course of the solution, it is probably always safe to arbitrarily replace this surd by 0 or anything else convenient, but until this step has been mathematically justified, it should be taken only when the greedy approach runs out of time or storage.

The quadratic final term of the above continued fraction illustrates another conjecture which, if true, would simplify the solution algorithm. We note that in converting a rational function to a continued fraction with Euclid's algorithm, most remainders are of degree one less than the corresponding divisor, and, consequently, the next partial quotient is linear. But if some remainder is "fortuitously" two or more powers less than the divisor, then the next quotient will be quadratic or greater. Recall that on the term preceding the quadratic (and last) term in our example, we were "fortuitously" able to annihilate three polynomial terms with two degrees of freedom. We therefore conjecture that the degree of a given polynomial is simply 1 + however many fortuitous annihilations occurred during the determination of the previous polynomial.

## Knowing When to Quit

How many terms of a continued fraction should we compute before relinquishing hope of its termination? I can only offer what *seems* to be a safe and reasonable bound, namely 1 + the sum of the magnitudes of the integer roots, $x_i$, of the resultant of $P(n)$ and $Q(n+x)$ with respect to $n$, where $P(n)$ and $Q(n)$ are the numerator and denominator of the term ratio $a_{n+1}/a_n$, and multiple roots are to be weighted by their multiplicities $m_i$. This represents all of the possible integer shifts of the denominator with respect to the numerator which result in one or more cancellations.

## Possible Extensions

Trigonometric sums might be handled by a process which first converts to complex exponential notation, then replaces some power of $e^{2iz}$ by the "base" $q$, thus forming a basic, or $q$ analog hypergeometric sum. Then we would apply the existing MACSYMA function to the corresponding ordinary hypergeometric, and form the $q$ analog of the result, if we got one. This is, however, highly speculative, and, in any event, would be unlikely to find such fancy telescoping functions as $f(n) = -1-\cos 2^n x$, which provides the identity

$$\sum_{n=0}^{m} \frac{1}{\sin 2^n x} = \cot \frac{x}{2} - \cot 2^m x .$$

Just as with definite integration, the problem of definite (typically infinite) summation is complicated by the bewildering variety of techniques available. One especially promising technique historically precedes and generalizes the method described in this paper (ref. 3). To see the relation between the methods, we point out another way of looking at the telescoping function $f(n)$, that is, as the "splitting function" determining the proportions into which the $n$th term of a series be partitioned, prior to combining the left portion of each term with the right portion of the preceding term. Writing $f_n$ for $f(n)$, we have

$$\begin{aligned} a_p + a_{p+1} + \ldots + a_q &= (-f_p+1+f_p)a_p + (-f_{p+1}+1+f_{p+1})a_{p+1} + \ldots + (-f_q+1+f_q)a_q \\ &= -a_p f_p + (1+f_p-f_{p+1}\frac{a_{p+1}}{a_p})a_p + \ldots + (1+f_{q-1}-f_q\frac{a_q}{a_{q-1}})a_{q-1} + (1+f_q)a_q , \end{aligned}$$

which yields equation (voila) upon the satisfaction of (func). But suppose it is not possible to annihilate the quantity

$$u_n = 1+f(n)-f(n+1)\frac{a_{n+1}}{a_n} = 1+f_n-f_{n+1}\frac{a_{n+1}}{a_n} .$$

Then we will have only succeeded in creating a new series whose terms are $u_n$ times the old ones. But if $u_n$ is reasonably simple and numerically small, it might be possible to iterate this splitting process indefinitely, so that in the limit, all of the original terms are multiplied by 0. When the various edge effects are taken into account, this process yields many interesting identities, such as

$$ {}_2F_1\left[{a,\ 1/2 \atop 3a};\ 3/4\right] = \frac{2^{4a+2}\, a!\, (3a)!}{3^{3a+1}\, (2a)!^2} . $$

Sometimes, the edge effects involve limits which have thus far eluded analysis, whereupon we invoke a nonrigorous technique which involves interpreting finite products over noninteger ranges. This results in conjectural identities such as

$$ {}_2F_1\left[{-a,\ 2a+1 \atop 2/3};\ 8/9\right] = 2\cdot 3^a \cos \pi(a+1/3) . $$

All of these conjectural formulas can be proven for countably many values of their parameters, and they have withstood extensive numerical testing at other values, but they remain tantalizingly uncertified.

Before the next MACSYMA Users' Conference, we hope to report on a partial implementation of a system for definite summation

## Late Developments

Kevin Karplus of Stanford has been developing a roughly parallel set of MACSYMA functions, so as to effectively double the rate of algorithmic experimentation. Discussions with him led me to discover that

$$ \sum_{n=0}^{m} n \frac{(n-1/2)!^2}{(n+1)!^2} = 4\pi - \frac{(12m+16)\,(m+1/2)!^2}{(m+1)!^2} $$

is an out-and-out counterexample to the greedy algorithm, since the correct telescoping function is

$$ f(n) = -12n^2 - 28n - 20 - \frac{4}{n} , $$

while the polynomial which most nearly satisfies (func) is

$$ p_1(n) = \frac{44n^2 + 60n + 4}{39} \quad (I) $$

As a result, I patched the algorithm to only determine $q$ of its $q+1$ undetermined coefficients on non terminal terms where $q > 1$, thus treating all such cases in the manner of (weirdo). This seemed to repair the problem, at the cost of exhausting list storage capacity on certain cases that had formerly worked. Fortunately, on 20 April 1977, all of this kludgery was rendered obsolete when I found a decision procedure for this problem. (A discrete analog to the Risch algorithm for indefinite integration.) The procedure is simpler, and makes better use of Jeff Golden's recently installed FUNMAKE and SUBST(LAMBDA( ... capabilities, and, as a result, runs ten to fifty times faster than the continued fraction algorithm. For those most interested, the details will be available in a handout at the conference.

Here follows the transcript of a short demo of both algorithms.

```
(C1) loadfile(bother,?>,dsk,rwg)$

BOTHER SUM15 DSK RWG being loaded
loading done
```

```
(C2) bothersum((-1)↑n/n↑2,n,1,m),%cfdepth:9;
TIME= 11585 MSEC.
                                      M
                                     ====       N
                                     \     (- 1)
(D2)                                  >    ------
                                     /        2
                                     ====    N
                                    N = 1
(C3) %cf;
TIME= 0 MSEC.                           1
                    ---------------------------------------- - 1
                                      1
                    ------------------------------------ - N
                                    2
                    -------------------------------- - N
                                  4
                    ---------------------------- - N
                                6
                    ------------------------ - N
                              9
                    -------------------- - N
                           12
                    ---------------- - N
                         16
                    ------------ - N
                       1
                    -------- - N
                    --MORE--
(D3)                --------------------------------------------
                                          2
```

**Old version fails (correctly) to find a closed form, but finds a nice continued fraction for $f(n)$, which it stores in %CF. Binding %CFDEPTH to an integer overrides the heuristic depth limiter.**

```
(C4) loadfile(nusum,?>,dsk,share)$

NUSUM 19 DSK SHARE ... loaded

(C5) n↑3*3↑n;
TIME= 3 MSEC.                          3  N
(D5)                                  N  3

(C6) nusum(%,n,0,m);
TIME= 1209 MSEC.
                            3      2                M
                      3 (4 M  - 6 M  + 12 M - 11) 3      3 11
(D6)                  ------------------------------ + ----
                                     8                   8
```

**New version (decision procedure) does an easy case.**

```
(C7) n↑3*3↑n*n!/prod(3*i-2,i,a,n+b);
TIME= 43 MSEC.
                                      3  N
                                     N  3  N!
(D7)                              ---------------
                                  N + B
                                  /===\
                                   ! !
                                   ! !  (3 I - 2)
                                   ! !
                                  I = A
(C8) nusum(%,n,0,m);
TIME= 5858 MSEC.
                                  2
                       3 (3 B - 2)  (3 B + 13)
(D8) ---------------------------------------------------------
                                                 B
                                               /===\
                                                ! !
     (3 B - 14) (3 B - 11) (3 B - 8) (3 B - 5)  ! !  (3 I - 2)
                                                ! !
                                               I = A

           3        2                 3        3        2                  2
 - 3 ((27 B  - 216 B  + 549 B - 440) M  + (81 B  - 486 B  + 945 B - 600) M

        3        2                      3      2                    M
 + (81 B  - 216 B  + 153 B - 30) M + 27 B  + 81 B  - 144 B + 52) 3  (M + 1)!

                                            M + B
                                            /===\
                                             ! !
/((3 B - 14) (3 B - 11) (3 B - 8) (3 B - 5)  ! !  (3 I - 2))
                                             ! !
                                            I = A

(C9) unsum(%,m);
TIME= 3005 MSEC.
                                     3  M
                                    M  3  M!
(D9)                             ---------------
                                 M + B
                                 /===\
                                  ! !
                                  ! !  (3 I - 2)
                                  ! !
                                 I = A

(C10)
```

**New version does a tougher case. UNSUM (backward difference) then checks it.**

## REFERENCES

1. Euler, L.: Consideratio Quarundam Serierum Quae Singularibus Proprietatibus Sunt Praeditae. *Novi Comentarii academiae scientarum Petropolitanae* 3 (1750/1), 1753, pp. 86-108

This appears in the Collected Works as

1a. *Opera Omnia*, Series I, Volume 14, Teubner, 1925, pp 525-6.

2. Chrystal, G.: *Algebra, an Elementary Text*, Chelsea, 1964, Chapter XXXI, Sec. 12, p. 392.

3. Gosper, R. Wm., Jr.: A Calculus of Series Rearrangements. *Algorithms and Complexity, New Directions and Recent Results*, Academic Press, 1976, (Ed. J. Traub), pp. 121-151.

This paper is stored as TELESC.XGP[1,RWG]@SAIL, converted from TELESC.POX by the POX document compiler, which was extended for this occasion by its developer, Robert Maas, of Stanford University.

BLANK PAGE

BLANK PAGE

# MODULAR POLYNOMIAL ARITHMETIC IN PARTIAL FRACTION DECOMPOSITION*

S. K. Abdali
B. F. Caviness
A. Pridor
Rensselaer Polytechnic Institute

## ABSTRACT

Algorithms for general partial fraction decomposition are obtained by using modular polynomial arithmetic. An algorithm is presented to compute inverses modulo a power of a polynomial in terms of inverses modulo that polynomial. This algorithm is used to make an improvement in the Kung-Tong partial fraction decomposition algorithm.

## INTRODUCTION

The partial fraction decomposition (pfd) of rational functions constitutes an important step in some symbolic integration algorithms (Horowitz ref. 1). Such a decomposition is frequently needed also in electrical network theory and control theory (e.g., Kuo ref. 2, Hsu and Meyer ref. 3). Consequently, a number of pfd algorithms dealing with the general and the important special cases (only linear or quadratic factors in the denominator of the rational function being decomposed) have appeared in the literature (see references in Kung and Tong, ref. 4). These algorithms fall into two categories: those based on applying the extended Euclidean algorithm (see Knuth ref. 5) and those based on solving linear systems of equations. Prior to 1969, the pfd algorithm most widely implemented in symbolic computation systems (e.g., Engelman's MATHLAB ref. 6, Moses' SIN ref. 7) was one of the former type and dated back to Hermite (ref. 8). Horowitz (ref. 1), however, discovered a faster algorithm of the latter type. The latter type algorithms require solving n linear equations in n unknowns, where n is the degree of the denominator in the rational fraction to be decomposed. Thus in the general case, they require $O(n^3)$ operations using classical elimination methods, or $O(n^{2.81})$ operations using Strassen's method (ref. 9). In special cases, the best bound is $O(n^2)$. But quite recently,

---

*Research partially supported by National Science Foundation Grant MCS-7623762.

Kung and Tong (ref. 4) have given an $O(n \log^2 n)$ algorithm which is again based on the extended Euclidean algorithm.

This paper uses the notation of modular polynomial arithmetic to derive pfd algorithms. This formulation brings out the similarities between the general pfd algorithms and the well-known technique of pfd by substitution for non-repeated linear factors (e.g., Kuo ref.2). The Kung-Tong algorithm is then easily derived as an adaptation of the general algorithm for fast computation. An algorithm is presented to obtain inverses modulo powers of a polynomial in terms of inverses modulo that polynomial. This is used in an improvement to Kung-Tong algorithm, which improvement although asymptotically minor, is believed to be of practical value in symbolic computation systems.

## PRELIMINARIES

Throughout this paper, polynomials are assumed to be univariate with coefficients in some given field.

Let B be a fixed polynomial. As usual, the relation congruence modulo B and the binary operation mod on polynomials are defined by

$$X \equiv Y \pmod{B} \quad \text{iff, for some polynomial } Q,\ X = QB + Y.$$

$$X \bmod B = Y, \text{ where } X \equiv Y \pmod{B} \text{ and } \deg(Y) < \deg(B).$$

Let A be a polynomial relatively prime to B. Then it is well-known (see, e.g., Herstein ref. 10) that there exist unique polynomials X, Y satisfying

$$AX + BY = 1\ ,\ \deg(X) < \deg(B),\ \deg(Y) < \deg(A). \qquad (1)$$

Accordingly we have the following:

Definition 2.1 (Inverse and division modulo B. Defined only if the denominator is relatively prime to B.)

(a) $\frac{1}{A} \bmod B = X$ where $AX \equiv 1 \pmod{B}$ and $\deg(X) < \deg(B)$

(b) $\frac{A}{C} \bmod B = (A \cdot (\frac{1}{C} \bmod B)) \bmod B$

Definition 2.2 (Truncated polynomial quotient)

$$\lfloor A/B \rfloor = (A - (A \bmod B))/B\ .$$

We use $M(n)$, $D(n)$, $F(n)$ to denote (an upper bound on) the number of operations needed, respectively, to multiply two polynomials of degree n, divide a polynomial of degree 2n by one of degree n, obtain polynomials X and Y of (1) when given A and B with max (deg (A), deg (B)) = n. We assume that the following convexity conditions are satisfied.

$$aM(n) \le M(an) \quad , \ a \ge 1$$

$$\Sigma M(n_i) \le M(\Sigma n_i) \quad , \quad n_i \text{ integer}$$

$$\Sigma F(n_i) \le F(\Sigma n_i) \quad , \quad n_i \text{ integer.}$$

It is reasonable to require such conditions as they are satisfied by the bounds M(n) and F(n) for all existing algorithms. Similar conditions are usually assumed, for example, by Aho, Hopcroft, and Ullman (ref. 11), Kung and Tong (ref. 4).

## PARTIAL FRACTION DECOMPOSITION PROBLEMS AND SIMPLE ALGORITHMS

Following Kung and Tong (ref. 4), we define three problems related to partial fraction decomposition.

1) General partial fraction decomposition (PF) Problem.

Let $Q_1,\ldots,Q_k$ be pairwise relatively prime polynomials of degree $n_1,\ldots,n_k$, respectively. Let $\ell_1,\ldots,\ell_k$ be positive integers and let P be a polynomial such that

$$\deg\ (P) < \deg\left(\prod_{i=1}^{k} Q_i^{\ell_i}\right) = \sum_{i=1}^{k} n_i \ell_i = n\ .$$

The problem is to obtain the polynomials $P_{ij}, 1 \le i \le k$, $1 \le j \le \ell_i$ satisfying

$$\frac{P}{\prod_{i=1}^{k} Q_i^{\ell_i}} = \sum_{i=1}^{k} \sum_{j=1}^{\ell_i} \frac{P_{ij}}{Q_i^j}\ , \ \deg\ (P_{ij}) < \deg\ (Q_i) = n_i\ . \quad (2)$$

$$(1 \le i \le k\ ,\ 1 \le j \le \ell_i),$$

2) Problem P1: (Special case of PF with $\ell_i = 1$, $1 \le i \le k$.)

Given pairwise relatively prime polynomials $R_1,\ldots,R_k$, and the polynomial P such that

$$\deg\ (P) < \deg\left(\prod_{i=1}^{k} R_i\right) .$$

The problem is to obtain the polynomials $C_1,\ldots,C_k$ satisfying

$$\frac{P}{\prod_{i=1}^{k} R_i} = \sum_{i=1}^{k} \frac{C_i}{R_i}\ , \ \deg(C_i) < \deg(R_i)\ ,\ 1 \le i \le k\,. \qquad (2)$$

3) Problem P2: (Special case of PF with k = 1.)

Given polynomials P, Q such that $\deg(P) < \deg(Q^\ell)$, to obtain the polynomials $C_1,\ldots,C_\ell$ satisfying

$$\frac{P}{Q^\ell} = \sum_{j=1}^{\ell} \frac{C_j}{Q^j}\ , \ \deg(C_j) < \deg(Q),\ 1 \le j \le \ell\,. \qquad (3)$$

It is well known (e.g., Horowitz ref. 1) that the polynomials to be determined in the above three problems all exist and are unique.

Using the modular polynomial arithmetic, we can now state simple algorithms for solving problems P1 and P2.

Algorithm 3.1 To solve P1.

```
for $i \leftarrow 1$ to $k$ do
    $C_i = \dfrac{P}{\left(\prod_{j=1}^{k} R_j\right)/R_i} \bmod R_i$ .
```

The algorithm is derived by multiplying both sides of (2) by $R_i$ and reducing each side modulo $R_i$.

Remark Note the similarity with the algorithm that works by substitution in the case of non-repeated linear factors (ref. 2). If $R_i = x - a$, then according to that algorithm one would obtain $C_i$ by substituting a for x in the fraction after cancelling x - a from the denominator. That is,

$$C_i = \left(\frac{P}{\frac{\prod R_j}{x-a}} \text{ evaluated with } x = a\right) = \frac{P}{\frac{\prod R_j}{x-a}} \bmod x - a\,.$$

Algorithm 3.1 is thus a straightforward generalization of that approach replacing substitutions by evaluation modulo a polynomial.

Algorithm 3.2 To solve P2.

```
begin $P' \leftarrow P$;
  for $j \leftarrow \ell$ downto 1 do
      begin
      $C_j \leftarrow P' \bmod Q$;
      $P' \leftarrow \lfloor P'/Q \rfloor$
      end
end
```

The PF problem can now be solved by cascading solutions of P1 and P2:

Algorithm 3.3 (Horowitz ref. 1) To solve PF.

```
begin
    compute $R_i \leftarrow Q_i^{\ell_i}$, $i = 1,\ldots,k$;
    solve problem P1 for $P/\prod_{i=1}^{k} R_i$, obtaining $C_i$ which
    satisfy (2);
    solve problems P2 for the fractions $C_i/Q_i^{\ell_i}$, $i=1,\ldots,k$;
end
```

The above algorithm lends itself to fast computation, and will be discussed further in section 5. We close this section with another useful algorithm which requires computing inverses modulo $Q_i$ only, not $Q_i^{\ell_i}$.

Algorithm 3.4 To solve PF.

```
begin
   $D \leftarrow P$; $E \leftarrow \prod_{i=1}^{k} Q_i^{\ell_i}$;
   for $i \leftarrow 1$ to $k$ do
       begin
       $E \leftarrow E/Q_i^{\ell_i}$;
       $F \leftarrow \frac{1}{E} \bmod Q_i$;
       for $j \leftarrow \ell_i$ downto 1 do
           begin
           $P_{ij} \leftarrow (D \cdot F) \bmod Q_i$;
           $D \leftarrow (D - P_{ij} \cdot E)/Q_i$
           end
       end
end
```

## COMPUTATION OF INVERSES MODULO A POWER OF A POLYNOMIAL

In this section, we consider the computation of $\frac{1}{A} \bmod B^{\ell}$, where A is relatively prime to B and deg (A) < $\ell$ deg (B). By applying, say, the Extended Euclidean Algorithm directly, we will need $O(F(\ell \cdot \deg(B)))$ operations. We describe below an alternative method in which use is made of the inverse modulo B only.

Lemma 4.1 Let A, B be relatively prime polynomials and let $X_i = \frac{1}{A} \bmod B^i$ for each i > 0. Then

$$X_{i+j} = (X_j + X_i(1 - AX_j)) \bmod B^{i+j} .$$

$$X_{2i} = (X_i \cdot (2 - AX_i)) \bmod B^{2i} .$$

These relations (with j = 1) are used below to compute $\frac{1}{A} \bmod B^{\ell}$ in a manner reminiscent of the binary algorithm for exponentiation (Knuth ref. 5).

Algorithm 4.1 Computation of $\frac{1}{A} \bmod B^{\ell}$

```
begin
$X_1 \leftarrow \frac{1}{A} \bmod B$; $D \leftarrow 1 - AX_1$ ;
$u \leftarrow \max\{2^w : 2^w \leq \ell\}$; $v \leftarrow \ell - u$; $z \leftarrow 1$; $C \leftarrow B$; $Z \leftarrow X_1$ ;
while $u > 1$ do
    begin
    $u \leftarrow u/2$;
    $q \leftarrow v \div u$;
    $C \leftarrow C^2$ ;
    $Z \leftarrow Z(2 - AZ) \bmod C$;
    $z \leftarrow 2z$;
    if $q = 1$ then
        begin
        $v \leftarrow v - u$;
        $C \leftarrow CB$; $Z \leftarrow (X_1 + 2D) \bmod C$;
        $z \leftarrow z + 1$;
        end
    end;
return z;
end
```

The correctness of the above algorithm follows from the fact that after execution of each line, it is the case that $zu + v = \ell$,

$0 \leq v < u$, $Z = X_z$, $C = B^z$, and u varies through consecutive decreasing powers of 2 from about $\ell$ at entry to 1 at exit (where $v = 0$).

Theorem 4.1 Algorithm 4.1 computes $\frac{1}{A} \bmod B^{\ell}$ in $O(F(\deg(B)) + (\log \ell)M(\ell \deg(B)))$ operations.

## FAST ALGORITHMS FOR PARTIAL FRACTION DECOMPOSITION

We now turn to the adaptation of the pfd methods for fast computations, the resulting algorithm being essentially that of Kung and Tong (ref. 4). In addition to the notation in the statement of the general PF problem, we use two other symbols:

$$\ell_{max} = \max(\ell_1, \ldots, \ell_k)$$

$$\bar{n} = n_1 + \ldots + n_k .$$

Lemma 5.1 Lines 1 and 3 of Algorithm 3.3 can be executed in $O(M(n))$ and $O((\log \ell_{max}) \cdot M(n))$ operations respectively.

The analysis of Line 2 of Algorithm 3.3 is more involved. This line requires the execution of Algorithm 3.1, i.e., the computation of

$$C_i = \frac{P}{\left(\prod_{j=1}^{k} R_j\right)/R_i} \bmod R_i , \quad 1 \leq i \leq k , \tag{1}$$

Writing R' for $\prod_{j=1}^{k} R_j$, we have

$$C_i = \frac{P}{\frac{R'}{R_i}} \bmod R_i = \left(\frac{P \bmod R_i}{\frac{R'}{R_i} \bmod R_i}\right) \bmod R_i . \tag{2}$$

The computation of all of $\frac{R'}{R_i} \bmod R_i$ is not easy to arrange for a fast algorithm. Instead, let us introduce the new quantity

$$R = \sum_{j=1}^{k} \frac{R'}{R_j} .$$

Now $R \bmod R_i = \left(\sum_{j=1}^{k} \frac{R'}{R_j}\right) \bmod R_i = \left(\frac{R'}{R_i} + \sum_{\substack{j=1\\ j\neq i}}^{k} \frac{R'}{R_j}\right) \bmod R_i = \frac{R'}{R_i} \bmod R_i$

since each term in the last summation is a multiple of $R_i$. Hence from (2), we get

$$C_i = \left(\frac{P \bmod R_i}{R \bmod R_i}\right) \bmod R_i .$$

(This result, derivable so readily in terms of modular arithmetic, has a more intricate proof in Kung and Tong (ref. 4)). That is,

$$C_i = \left((P \bmod R_i)\left[\frac{1}{R \bmod R_i} \bmod Q_i^{\ell_i}\right]\right) \bmod R_i . \qquad (4)$$

By using a binary splitting technique, Kung and Tong (ref. 4) show how to obtain all of $P \bmod R_i$ and $R \bmod R_i$ in $O((\log \kappa)\cdot M(n))$ operations. For the inverse part we may use Algorithm 4.1. Hence by Theorem 4.1 and the assumptions on the bound $F(n)$, we obtain

Lemma 5.2 All of the inverses in (5) can be computed in $F(\bar{n}) + O((\log \ell_{max})\cdot M(n))$ operations.

Now we have

Lemma 5.3 Line 2 of Algorithm 3.3 can be executed in $F(\bar{n}) + O((\log \ell_{max})\cdot M(n)) + O((\log k)\cdot M(n))$ operations.

Theorem 5.1 The general PF problem can be done in $F(\bar{n}) + O((\log \ell_{max})\cdot M(n)) + O((\log k)\cdot M(n))$ operations.

The original Kung-Tong algorithm requires $F(n)$ instead of $F(\bar{n})$ as the first term. Recall that $n = \Sigma n_i \ell_i$, while $\bar{n} = \Sigma n_i$.

# REFERENCES

1. Horowitz, E.: Algorithms for Symbolic Integration of Rational Functions, Ph.D. dissertation, Univ. of Wisconsin, Madison, Wisconsin, 1969. See also Algorithms for Partial Fraction Decomposition and Rational Function Integration, in Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, S. R. Petrick ed. (March 1971), ACM, New York.

2. Kuo, F. F.: Network Analysis and Synthesis, 2nd edition, John Wiley, New York, 1966.

3. Hsu, J. C.; Meyer, A. U.: Modern Control Principles and Applications, McGraw-Hill, New York, 1968.

4. Kung, H. T.; Tong, D. M.: Fast Algorithms for Partial Fraction Decomposition, Tech. Report, Computer Science Department, Carnegie-Mellon University, February 1976.

5. Knuth, D. E.: Semi-Numerical Algorithms, Addison-Wesley, Reading, Mass., 1969.

6. Engelman, C.: MATHLAB: A Program for On-Line Assistance in Symbolic Computations, Proc. 1965 FJCC, Spartan Books, Washington, D. C.

7. Moses, J.: Symbolic Integration, Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, Mass., 1967.

8. Hermite, C.: Oeuvres de Charles Hermite, (Picard E.: editor), Vol. 3, Gauthiers-Villars, Paris, 1912.

9. Strassen, V.: Gaussian Elimination Is Not Optimal, Numerische Math., 13, 1969, pp. 354-356.

10. Herstein, I. N.: Topics in Algebra, Blaisdell Publishing Co., New York, 1965.

11. Aho, A. V.; Hopcroft, J. E.; Ullman, J. D.: The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974.

BLANK PAGE

BLANK PAGE

# A NEW ALGORITHM FOR THE INTEGRATION OF EXPONENTIAL AND LOGARITHMIC FUNCTIONS*

Michael Rothstein
Universidad Simon Bolivar

## ABSTRACT

A new algorithm for symbolic integration of functions built up from the rational functions by repeatedly applying either the exponential or logarithm functions is discussed. This new algorithm does not require polynomial factorization nor partial fraction decomposition and requires solutions of linear systems with only a small number of unknowns. It is proven that if this algorithm is applied to rational functions over the integers, a computing time bound for the algorithm can be obtained which is a polynomial in (1) a bound on the integer length of the coefficients, and (2) the degrees, of the numerator and denominator of the rational function involved.

## INTRODUCTION AND SOME NECESSARY CONCEPTS

In this paper we discuss a new algorithm for symbolic integration of rational functions of logarithms and exponentials obtained (roughly speaking) by repeatedly applying the logarithm and exponential functions to rational functions in the integration variable. No restriction is placed on the constant field, except that arithmetic in this field be recursive, and that no functional expression obtainable from our expressions above by addition, subtraction, multiplication and division be a new constant.

As many authors have done in this area (see a complete history of the subject in ref. 1) we shall use the notation and concepts described by Risch (ref. 2). In particular we shall work with differential fields of the form

$$F = F_n = K(z, \theta_1, \ldots, \theta_n)$$

where K is the constant field of F, z is the integration variable,

---

*Work supported in part by National Science Foundation Grant MCS76-23762 (to Rensselaer Polytechnic Institute) and by Grants GJ 32181 and MCS76-15035 (to University of Utah).

and each $\theta_n$ is a monomial (logarithmic or exponential) over

$$F_{i-1} = K(z,\theta_1,\ldots,\theta_{i-1}) \text{ , } F_o = K(z) \text{ .}$$

We shall also say that $F_n$ is a Liouville extension of $F_i$ ($i < n$) in this situation.

Our algorithm will require the existence of algorithms to perform arithmetic in K, and also, algorithms for the usual arithmetic operation defined on the domains $S_i = F_{i-1}[\theta_i]$ and $E_i = \{P/\theta_i^{\ell}, P \in S_i \text{ and } \ell \in Z\}$, like addition, subtraction, multiplication, and division (for elements of $S_i$, obtaining a quotient and a remainder).

Finding gcd's (greatest common divisors) of elements of $S_i$ can be done by applying Euclid's algorithm. Notice that this gcd is always monic. For $E_i$, we define the gcd of two elements f and g by pointing out that we can find P and Q in $S_i$ such that gcd $(P,\theta_i)$ = gcd $(Q,\theta_i) = 1$ and for some integers j, m, we have that $f = P\theta_i^j$ and $g = Q\theta_i^m$ . We then define gcd (f,g) = gcd (P,Q).

We shall also require algorithms for finding X and Y in $S_i$ such that $AX + BY = C$ with deg $X <$ deg B for given A, B, C in $S_i$ with gcd $(A,B) = 1$. We shall refer to these equations as univariate polynomial equations (U.P.E.'s).

Finally, we will need the ability to compute the resultant of given elements A, B of $S_i[\alpha]$ (where $\alpha$ is some indeterminate over $S_i$) with respect to $\theta_i$. We shall denote this function by Res $(A,B,\theta_i)$.

Now some more definitions:

a) Given a non-zero element f of $F_m$ ($m \leq n$) there exist unique P, Q in $S_m$ such that $P/Q = f$, gcd$(P,Q) = 1$ and Q is monic. We shall call P the numerator (denoted by num f) and Q the denominator (denoted by den f) of f. Let us also define num 0 = 0 and den 0 = 1.

b) We shall say that f in $F_m$ is a proper element of $F_m$ if $f = 0$ or deg (num f) $<$ deg (den f) and also, if $\theta_m$ is exponential over $F_{m-1}$, then $\theta_m$ does not divide den f. This implies that all square free factors q of den f satisfy gcd$(q,q') = 1$.

c) If f is a proper element of $F_m$, we shall say that f is normal (in $F_m$) if den f is square-free (equivalently, if gcd (den f, (den f)') = 1).

d) Let $D_m$ denote $E_m$ if $\theta_m = \exp u$, $u \in F_{m-1}$, otherwise $D_m = S_m$.

Notice that all these definitions are valid with m = 0 and $F_{-1} = K$.

## ALGORITHM OUTLINE

We shall now discuss the operations done by our algorithm when presented with some integrand $f(z) \in F_n$. Let Q = num f, R = den f and, by a division process, obtain $P_1$, T in $S_n$ such that

$$Q = P_1 R + T \text{ , deg } T < \text{deg } R \text{ , or } T = 0 \text{ .}$$

If $\theta_n$ is not exponential over $F_{n-1}$, we now have to compute

$$\int P_1 \quad \text{and} \quad \int \frac{T}{R} \text{ .}$$

Otherwise, let $R = \theta_n^j R_1$, $R_1$ in $S_n$, gcd $(R_1, \theta_n) = 1$, and solve the U.P.E.

$$T = \theta_n^j T_1 + R_1 T_2$$

for $T_1$, $T_2$, with deg $T_1$ < deg $R_1$ (and deg $T_2$ < j).

We then have to compute

$$\int f(z) = \int P_1 + \int \frac{T}{R} = \int (P_1 + \frac{T_2}{\theta_n^j}) + \int \frac{T_1}{R_1}$$

and thus, we have succeeded in decomposing our integral into integrating an element of $D_n$ and integrating a proper element of $F_n$.

To integrate elements of $D_n$, we employ a method similar to one described by Risch (ref. 2) with the following changes:

a) In the logarithmic case, the algorithm invoked recursively is the algorithm described herein instead of Risch's. A special purpose algorithm is also discussed in reference 1, pp. 46-49.

b) In the exponential case, we use a different algorithm to solve the resulting differential equation for X

$$X' + u'X = T$$

with X, u, T in $F_{n-1}$, where exp (u) is a regular monomial over $F_{n-1}$.

This algorithm will be described in section 3.

To integrate proper elements g of $F_n$ we use an algorithm described by D. Mack (ref. 3) which yields

$$\int g = h_1 + \int h_2$$

where $h_2$ is normal in $F_n$. Our algorithm to find $\int h_2$ will be described in section 4. In section 5 we will present a computing time analysis for the rational function case.

## SOLVING A SPECIAL CASE OF A DIFFERENTIAL EQUATION

In this section we will present a method for solving the differential equation

$$X' + vX = T \tag{1}$$

for X in $F_n$, assuming that v, T are in $F_n$, and that $\exp(\int v)$ is a regular monomial over $F_n(\int v)$, where $\int v$, if not in $F_n$, is elementary over $F_n$. Thus, X, if it exists, is unique.

(The reason for not requiring $\int v$ to be in $F_n$ is that this algorithm will be invoked recursively and under those circumstances, we cannot guarantee that $\int v$ be in $F_n$, even though our other conditions will apply.)

In order to find X, let

$$v = \frac{v_1}{v_2} \qquad \text{and} \qquad T = \frac{T_1}{T_2}$$

with $v_1$, $T_1$ in $D_n$, $v_2$, $T_2$ in $S_n$, where, if $\theta_n$ is exponential over $F_{n-1}$ $(D_n = E_n)$ then $\theta_n \nmid v_2$, and $\theta_n \nmid T_2$. We will also require that gcd $(v_1,v_2)$ and gcd $(T_1,T_2) = 1$ and that $v_2,T_2$ be monic.

Let us further factor

$$v_1 = \bar{v}\,\hat{v} \qquad \text{and} \qquad T_1 = \bar{T}\,\hat{T}$$

in such a way that $\hat{v}$, $\hat{T} \in S_n$ are monic, gcd $(\bar{v},T_2)$ = gcd $(v_2,\bar{T})$ = 1, $\bar{v}$, $\bar{T}$ are in $D_n$ and every square-free factor of $\hat{v}$ (respectively $\hat{T}$) divides $T_2$ (respectively $v_2$). We can then prove that gcd $(\bar{v},\hat{v})$ = gvd $(\bar{T},\hat{T}) = 1$.

Now, let $p_1,\ldots,p_k$ be a square-free basis for $\hat{v}$, $\hat{T}$, $v_2$, $T_2$. Assume each $p_i$ is monic, obtaining

$$v = \frac{\bar{v}}{\prod_{i=1}^{k} p_i^{b_i}} \qquad\qquad T = \frac{\bar{T}}{\prod_{i=1}^{k} p_i^{c_i}}$$

where the $b_i,c_i$ are integers with $b_i \neq 0$ if $c_i = 0$.

It can be shown that X can then be represented uniquely as

$$X = \frac{\bar{X}}{\prod_{i=1}^{k} p_i^{x_i}} \qquad \text{with } \bar{X} \in D_n$$

if we assume that no $p_i$ divides $\bar{X}$ (though possibly gcd $(\bar{X},p_i) \neq 1$).

We will now find the $x_i$, as follows: If we substitute these values of X, v, T in (1), we obtain

$$\frac{\bar{X}'\prod_{i=1}^{k} p_i - \bar{X}\sum_{i=1}^{k} \left(x_i p_i' \sum_{\substack{j=1\\ j\neq i}}^{k} p_j\right)}{\prod_{i=1}^{k} p_i^{x_i+1}} + \frac{\bar{v}\,\bar{X}}{\prod_{i=1}^{k} p_i^{x_i+b_i}} = \frac{\bar{T}}{\prod_{i=1}^{k} p_i^{c_i}} .$$

If $b_i \neq 1$ we notice that $x_i = c_i - \max(b_i, 1)$. Otherwise (for $b_{i_o} = 1$) we can have $p_{i_o}$ dividing the numerator of this expression. But this can happen if and only if

$$\gcd\left[p_{i_o},\ \theta_n^{\alpha}\left(\bar{v}\prod_{i=1}^{k} p_i^{a_i-b_i} - x_{i_o} p'_{i_o} \prod_{\substack{i=1\\ i\neq i_o}}^{k} p_i^{a_i}\right)\right] \neq 1$$

where $a_i = \max(b_i, 1)$ and $\alpha$ is the smallest non-negative integer such that the expression in parenthesis times $\theta_n^{\alpha}$ belongs to $S_n$.

But this is true if and only if

$$\operatorname{Res}\left(p_{i_o}, \bar{v}\,\theta_n^{\alpha}\prod_{i=1}^{k} p_i^{a_i-b_i} - x_{i_o}\,\theta_n^{\alpha} p'_{i_o} \prod_{\substack{i=1\\ i\neq i_o}}^{k} p_i^{a_i},\ \theta_n\right) = 0.$$

Since this is a polynomial equation in $x_{i_o}$, we can check whether our root is an integer (bigger than $c_{i_o} - 1$) and solve for it;

otherwise we set $x_{i_o} = c_{i_o} - 1$.

We then obtain an equation of the form

$$\bar{A}\,\bar{X}' + \bar{B}\,\bar{X} = \bar{C}$$

with $\bar{A}$, $\bar{B}$, $\bar{C}$, $\bar{X}$ in $D_n$. If $\theta_n$ is exponential over $F_{n-1}$ we can do a similar computation to find an equivalent equation with $\bar{A}$, $\bar{B}$, $\bar{C}$, $\bar{X}$ in $S_n$. Thus, assume $\bar{A}$, $\bar{B}$, $\bar{C}$, $\bar{X}$ are in $S_n$. In order to find $\bar{X}$, we now do the following analysis:

We can assume that gcd $(\bar{A}, \bar{B}) = 1$, since otherwise, let $g = \gcd(\bar{A}\ \bar{B})$. Then $g|\bar{C}$ (otherwise no solution exists) and we obtain the equivalent equation $\frac{\bar{A}}{g}\bar{X}' + \frac{\bar{B}}{g}\bar{X} = \frac{\bar{C}}{g}$.

We have three different cases:

i) deg $\bar{A} = 0$ and deg $\bar{B} > 0$.

In this case, either $\bar{C} = 0$, (so that $\bar{X} = 0$) or deg $\bar{B} >$ deg $\bar{C}$, so that no solution exists, or deg $\bar{B} \leq$ deg $\bar{C}$ and we can find the leading coefficient of $\bar{X}$, (since deg $\bar{A}\,\bar{X}' <$ deg $\bar{B}\,\bar{X}$) arriving at an equation of the form

$$\bar{A}\,\hat{X}' + \bar{B}\,\hat{X} = \hat{C}$$

with deg $\hat{C} <$ deg $\bar{C}$, so that we can solve it recursively.

ii) deg $\bar{A}$ = deg $\bar{B} = 0$ .

Since, by assumption, our solution, if it exists, is unique, we obtain that deg $\bar{X}$ = deg $\bar{C}$ and a set of equations of the form (1) but with $v$, $T$, $X$ in $F_{n-1}$. We then enter our algorithm recursively, noting (though not trivially) that these equations satisfy the same conditions we had before, with respect to $v$.

iii) deg $\bar{A} > 0$.

In this case, we point out that if we let $\bar{X} = Q\bar{A} + R$, ($\deg R < \deg \bar{A}$) and substitute, we obtain

$$\bar{C} = \bar{A}\,\bar{X}' + \bar{B}\,\bar{X} = \bar{A}(Q'\bar{A} + Q(\bar{A}' + \bar{B}) + R') + \bar{B}R .$$

Thus if we solve the UPE

$$Y\bar{A} + Z\bar{B} = \bar{C} \quad \text{for } Y \text{ and } Z$$

with $\deg Z < \deg \bar{A}$, we must have that $Z = R$, and $Q$ must be the solution of the equation

$$\bar{A}Q' + (\bar{B} + \bar{A}')Q = Y - Z'$$

which we can solve (if $Y = Z'$ then $Q = 0$) by checking gcd $(\bar{A}, \bar{B}+\bar{A}')$ and applying one of (i), (ii) or (iii) again.

It is very important to note that (iii) should be applied after computing a bound on $\deg \bar{X}$ and noting that $\deg Q \leq \deg \bar{X} - \deg \bar{A}$. If we obtain that $\deg Q < 0$, then there is no possible solution. Note that after the first time we apply (iii), no computation on the bound of $X$ is required, since this bound is already known. Finally, the first time we apply (iii), we compute a bound on $\deg X$ using methods described in reference 2.

## INTEGRATION OF NORMAL ELEMENTS OF $F_n$

In this section we will present a new algorithm for finding the integral of a normal element of $F_n$. The algorithm is justified and explained in the following:

Theorem 1.

Let $f$ be normal in $F_n$, $P = \text{num } f$, $Q = \text{den } f$. Let $r(\alpha) =$ resultant $(P - \alpha Q', Q, \theta_n)$. Then $\int f$ is elementary if and only if all the roots of $r(\alpha)$ are constants, if and only if $r(\alpha) = s\,t(\alpha)$ with $t(\alpha) \in K[\alpha]$ and $s \in S_n$.

Theorem 2.

Using the same notation as in Theorem 1, if $f$ is elementary, let $c_1, \ldots, c_m$ be the roots of $r(\alpha)$ and $v_i = \gcd(P - c_iQ', Q)$. Then

i) If $\theta_n$ is logarithmic over $F_{n-1}$ or $n = 0$, then

$$f = \sum_{i=1}^{m} c_i \frac{v_i'}{v_i} .$$

ii) If $\theta_n = \exp(w)$, $w \in F_{n-1}$, then $f = \sum_{i=1}^{m} c_i (\frac{v_i'}{v_i} - n_i w')$

where $n_i = \deg v_i$ .

Theorem 3.

Using the same notation as in the two previous theorems, if f is elementary, then $r(\alpha)$ (and $t(\alpha)$) define the least degree extension of the constant field, necessary to express the integral of f. This theorem answers affirmatively the open problem asked by Risch on page 171 of reference 2, and generalizes a result of Trager (ref. 4). For proofs of these statements, we refer the reader to reference 1.

## COMPUTING TIME ANALYSIS FOR THE RATIONAL FUNCTION CASE

In this section, we will present a computing time analysis of this algorithm for the rational function case. First if P is a polynomial with integer coefficients,

$P = \sum_{i=0}^{n} a_i x^i$, we define

$$\text{norm } P = |P| = \sum_{i=0}^{n} |a_i| .$$

Now, we define $F(m,n,d)$ as the class of functions $P/Q$, with P, Q relatively prime univariate polynomials over the integers, $\max(|P|,|Q|) \leq d$ , $\deg P \leq m$ , $\deg Q \leq n$ .

We shall use the definitions and notation for dominance and codominance used, for example, by Collins (ref. 5).

Then, we have the following theorem. For $f \in F(m,n,d)$, the time required by the algorithm described herein is given by

$$T_{INTG}(m,n,d) \preceq n^8 L^2(dn) + n^6 L^3(kn)$$

# AUTHOR INDEX

Abdali . . . 253 3/F10
Ament . . . 87 2/A7
Andersen . . . 161 2/F11
Anderson . . . 395 5/C8
Avgoustis . . . 21 1/C6
Bogen . . . 11, 75 1/B10
Bulnes . . . 461 6/A9
Caviness . . . 253 3/F10
Char . . . 53 1/E10
Cohen . . . 275 4/A9
Cuthill . . . 131 2/E3
Doohovskoy . . . 473 6/B7
Fateman . . . 43, 327 1/D14
Fennelly . . . 97 2/B3
Geddes . . . 405 5/D4
Genesereth . . . 291, 309 4/B11
Golden . . . 1, 109 1/A14
Gosper . . . 237 3/E8
Grünbaum . . . 491 6/C11
Gupta . . . 151 2/F1
Ivie . . . 317 4/D9
Karney . . . 377 5/B4
Kulp . . . 385 5/B12
Lafferty . . . 347 4/F11
Lau . . . 395 5/C8
Lewis . . . 277 4/A11
Moenck . . . 225 3/D10
Moses . . . 123, 275 2/D1
Ng . . . 151, 177 2/F1
Noor . . . 161 2/F11
Pavelle . . . 75, 97 1/G4
Poole . . . 145 2/E9
Pridor . . . 253 3/F10
Rothstein . . . 263 3/G6
Spear . . . 369 5/A10
Steele . . . 203, 215 3/C2
Stoutemyer . . . 315, 425, 447 4/D7
Wahlquist . . . 71 1/F14
Wang . . . 55, 435 1/E12
Yun . . . 65 1/F8
Zippel . . . 361 4/G11

# CONTENTS

FOREWORD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iii **1/A1**

PREFACE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . v **1/A12**

AUTHOR INDEX . . . . . . . . . . . . . . . . . . . . . . . . . . . . vii **1/A1**

1. MACSYMA'S SYMBOLIC ORDINARY DIFFERENTIAL EQUATION SOLVER . . . . . . . 1 **1/A14**
   Jeffrey P. Golden

2. A PROGRAM FOR THE SOLUTION OF INTEGRAL EQUATIONS . . . . . . . . . . . 11 **1/B10**
   Richard A. Bogen

3. SYMBOLIC LAPLACE TRANSFORMS OF SPECIAL FUNCTIONS . . . . . . . . . . . 21 **1/C6**
   Yannis Avgoustis

4. AN IMPROVED ALGORITHM FOR THE ISOLATION OF POLYNOMIAL REAL ZEROS . . . 43 **1/D1**
   Richard J. Fateman

5. FLOATING POINT ISOLATION OF ZEROS OF A REAL POLYNOMIAL VIA MACSYMA . . 53 **1/E1**
   Bruce W. Char

6. PRESERVING SPARSENESS IN MULTIVARIATE POLYNOMIAL FACTORIZATION . . . . 55 **1/E12**
   Paul S. Wang

7. ON THE EQUIVALENCE OF POLYNOMIAL GCD AND SQUAREFREE FACTORIZATION PROBLEMS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65 **1/F8**
   David Y. Y. Yun

8. DIFFERENTIAL FORM ANALYSIS USING MACSYMA . . . . . . . . . . . . . . . 71 **1/F14**
   Hugo D. Wahlquist

9. INDICIAL TENSOR MANIPULATION ON MACSYMA . . . . . . . . . . . . . . . . 75 **1/G4**
   Richard A. Bogen and Richard Pavelle

10. PURE FIELD THEORIES AND MACSYMA ALGORITHMS . . . . . . . . . . . . . . 87 **2/A7**
    William S. Ament

11. BLACK HOLES AND RELATIVISTIC GRAVITY THEORIES . . . . . . . . . . . . . 97 **2/B3**
    A. J. Fennelly and Richard Pavelle

12. THE EVALUATION OF ATOMIC VARIABLES IN MACSYMA . . . . . . . . . . . . . 109 **2/C1**
    Jeffrey P. Golden

13. THE VARIETY OF VARIABLES IN MATHEMATICAL EXPRESSIONS . . . . . . . . . 123 2/D1
Joel Moses

14. RATIONAL APPROXIMATION TO $e^{-x}$ WITH NEGATIVE REAL POLES . . . . . . . . 131 2/D9
Elizabeth Cuthill

15. TIMING FORMULAS FOR DISSECTION ALGORITHMS ON VECTOR COMPUTERS . . . . 145 2/E9
W. G. Poole, Jr.

16. SYMBOLIC CALCULATIONS IN A FINITE DYNAMIC ELEMENT ANALYSIS . . . . . 151 2/F1
Kajal K. Gupta and Edward W. Ng

17. SYMBOLIC MANIPULATION TECHNIQUES FOR VIBRATION ANALYSIS OF LAMINATED ELLIPTIC PLATES . . . . . . . . . . . . . . . . . . . . . . . 161 2/F11
C. M. Andersen and Ahmed K. Noor

18. OBSERVATIONS ON APPROXIMATE INTEGRATIONS . . . . . . . . . . . . . . . 177 2/G13
Edward W. Ng

19. LISP: PROGRAM IS DATA – A HISTORICAL PERSPECTIVE ON MACLISP . . . . 181 3/A8
Jon L White

20. LISP: DATA IS PROGRAM – A TUTORIAL IN LISP . . . . . . . . . . . . . . 191 3/B4
Jon L White

21. DATA REPRESENTATIONS IN PDP-10 MACLISP . . . . . . . . . . . . . . . . 203 3/C2
Guy Lewis Steele Jr.

22. FAST ARITHMETIC IN MACLISP . . . . . . . . . . . . . . . . . . . . . 215 3/C14
Guy Lewis Steele Jr.

23. ON COMPUTING CLOSED FORMS FOR SUMMATIONS . . . . . . . . . . . . . . 225 3/D10
Robert Moenck

24. INDEFINITE HYPERGEOMETRIC SUMS IN MACSYMA . . . . . . . . . . . . . . 237 3/E8
R. Wm. Gosper, Jr.

25. MODULAR POLYNOMIAL ARITHMETIC IN PARTIAL FRACTION DECOMPOSITION . . . 253 3/F10
S. K. Abdali, B. F. Caviness, and A. Pridor

26. A NEW ALGORITHM FOR THE INTEGRATION OF EXPONENTIAL AND LOGARITHMIC FUNCTIONS . . . . . . . . . . . . . . . . . . . . . . . . 263 3/G6
Michael Rothstein

27. SUMMATION OF RATIONAL EXPONENTIAL EXPRESSIONS IN CLOSED FORM . . . . 275 4/A9
Joel Moses and Jacques Cohen

28. USER AIDS FOR MACSYMA . . . . . . . . . . . . . . . . . . . . . . . . 277 4/A11
V. Ellen Lewis

29. THE DIFFICULTIES OF USING MACSYMA AND THE FUNCTION OF USER AIDS . . . 291 4/B11
Michael R. Genesereth

30. AN AUTOMATED CONSULTANT FOR MACSYMA . . . . . . . . . . . . . . . . . . 309 4/D1
Michael R. Genesereth

31. A MACSYMA COMPUTER-ALGEBRA MOVIE DEMONSTRATION . . . . . . . . . . . 315 4/D7
David R. Stoutemyer

32. SOME MACSYMA PROGRAMS FOR SOLVING DIFFERENCE EQUATIONS . . . . . . . 317 4/D9
John Ivie

33. SOME COMMENTS ON SERIES SOLUTIONS . . . . . . . . . . . . . . . . . . . 327 4/E5
Richard J. Fateman

34. POWER SERIES SOLUTIONS OF ORDINARY DIFFERENTIAL EQUATIONS IN
MACSYMA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 347 4/F11
Edward L. Lafferty

35. RADICAL SIMPLIFICATION MADE EASY . . . . . . . . . . . . . . . . . . . 361 4/G11
Richard E. B. Zippel

36. A CONSTRUCTIVE APPROACH TO COMMUTATIVE RING THEORY . . . . . . . . . 369 5/A10
David A. Spear

37. REDUCTION OF THE EQUATION FOR LOWER HYBRID WAVES IN A PLASMA
TO A NONLINEAR SCHRÖDINGER EQUATION . . . . . . . . . . . . . . . . 377 5/B4
Charles F. F. Karney

38. RAY TRAJECTORIES IN A TORUS: AN APPLICATION OF MACSYMA TO A COMPLEX
NUMERICAL COMPUTATION . . . . . . . . . . . . . . . . . . . . . . . 385 5/B12
John L. Kulp

39. APPLICATION OF MACSYMA TO FIRST ORDER PERTURBATION THEORY IN
CELESTIAL MECHANICS . . . . . . . . . . . . . . . . . . . . . . . . 395 5/C8
John D. Anderson and Eunice L. Lau

40. SYMBOLIC COMPUTATION OF RECURRENCE EQUATIONS FOR THE CHEBYSHEV
SERIES SOLUTION OF LINEAR ODE's . . . . . . . . . . . . . . . . . . 405 5/D4
K. O. Geddes

41. sin(x)**2 + cos(x)**2 = 1 . . . . . . . . . . . . . . . . . . . . . . . 425 5/E10
David R. Stoutemyer

42. MATRIX COMPUTATIONS IN MACSYMA . . . . . . . . . . . . . . . . . . . . 435 5/F6
Paul S. Wang

43. SYMBOLIC COMPUTER VECTOR ANALYSIS . . . . . . . . . . . . . . . . . . 447 5/G4
David R. Stoutemyer

44. A NATURAL WAY TO DO SPATIAL LINEAR GEOMETRY IN MACSYMA . . . . . . . 461 6/A9
Juan Bulnes

45. VARIETIES OF OPERATOR MANIPULATION . . . . . . . . . . . . . . . . . 473 6/B7
Alexander Doohovskoy

46. PROGRESS REPORT ON THE DETERMINANT OF A RANDOM MATRIX . . . . . . . . 491 6/C11
F. A. Grünbaum

BIBLIOGRAPHY . . . . . . . . . . . . . . . . . . . . . . . . . . . . 493 6/C13

$$+ \max(m + 1 - n, 0)nL^2(d) + 1$$

(if we assume that the norm of any of the partial results except the resultant, is also bounded by d) where $L(d) = \log_2(d) + 1$.

Proof: We have two cases to consider.

(a) $m \geq n$, and

(b) $m < n$.

If $m < n$, we do a quotient-remainder operation, and then we continue with D. Mack's algorithm and the algorithm described in section 4. We then have the following computing times.

The quotient-remainder operation requires constant time. D. Mack's algorithm requires time $n^5L(nd)^2$ as proven in reference 3.

The algorithm in section 4 requires time dominated by:

i) $nL(d)$ to compute $Q'$

ii) $nL(d)$ to compute $P - \alpha Q'$ ($\deg P < n$)

iii) $n^3L(d)$ to compute $R = \text{resultant}(P - \alpha Q', Q)$.

(We point out that $\deg_\alpha R \leq n$, and its norm is bounded by $(2n)!d^{2n} \leq 2n^{2n}d^{2n} = (2dn)^{2n}$ and thus $L(\text{norm } R) \preceq nL(dn)$).

iv) $n^8 + n^6L^2(\text{norm } R) + n^3L^3(\text{norm } R) \preceq n^8L^2(dn) + n^6L^3(dn)$ to compute the roots of R (as given in private communication from G.E. Collins assuming number of roots = n).

v) $n(n^2L(d) + nL^2(d))$ to compute $\gcd(P - c_iQ', Q)$ for $1 \leq i \leq n$ (assuming there are n distinct roots of R).

Adding these times, it is clear that the time to compute the roots of R dominates all other computing times, and we obtain the desired result that the computing time for the algorithm in section 4 is dominated by

$$n^8L^2(dn) + n^6L^3(dn).$$

Finally, if $m > n$, the time to compute the quotient-remainder is given by $(m + 1 - n)nL^2(d)$ and the time to compute the integral of the polynomial part (by the classical method) is given by $(m + 1 - n)L^2(d)$.

If we add all these computing times we obtain the result that we quoted at the beginning.

Note: The bounds on the time to compute the resultant and the norm of R were obtained from reference 5.

## CONCLUSIONS

We have shown that for rational functions integration in finite terms can be done in time bounded by a polynomial in the size of the input, if part of that size is the degree.

In the general case, we conjecture that the computing time, for the case where the number of monomials is fixed, yields a polynomial in the same sense as above. (No better bound can be obtained, as shown by the example $\int x^n e^x dx$.) This conjecture, though, implies that the computing time of any algorithm for symbolic integration is at least exponential in the number of monomials in the integrand.

## REFERENCES

1. Rothstein, Michael: Aspects of Symbolic Integration and Simplification of Exponential and Primitive Functions, Ph.D. Thesis, University of Wisconsin, Madison, 1976. Available from Xerox University Microfilms, Ann Arbor, Michigan.

Risch, Robert H.: The Problem of Integration in Finite Terms, Trans. Amer. Math. Soc., 139, May 1969, pp. 167-189.

Mack, Dieter: On Rational Integration, University of Utah, Computational Physics Group Report UCP-38, September 1975.

4. Trager, Barry M.: Algebraic Factoring and Rational Function Integration, Proceedings of the 1976 Symposium on Symbolic and Algebraic Computation, R. D. Jenks, Ed., Assoc. for Comp. Mach. N. Y., 1976.

5. Collins, George E.: The Calculation of Multivariate Polynomial Resultants, JACM 18, #4, October 1971, pp. 515-532.

# SUMMATION OF RATIONAL EXPONENTIAL EXPRESSIONS IN CLOSED FORM

Joel Moses*
and Jacques Cohen**

## ABSTRACT

A program is described which provides, whenever possible, symbolic closed form solutions to summations of rational exponential expressions, i.e., of the type

$$\sum_{x=\ell}^{x=u} (F_0 + F_1/F_2)a^x$$

where the F's are polynomials in x. The program is based on a decision procedure recently developed by M. Karr. The decision procedure consists of determining if the resulting sum is in itself a rational exponential, and if so, generating that expression. The paper first reviews some of the classical techniques summarized by G. Boole for attempting to find closed forms for the given type of summations. Karr's method is then informally presented. His method not only provides a decision procedure but also appears better suited for computer implementation than the classical techniques. Several examples of the program's use are provided.

---

*. Laboratory for Computer Science, M.I.T., Cambridge, Mass. 02139. This work was supported by the United States Energy Research and Development Administration under contract E(11-1)-3070 and by NASA under Grant NSG 1323.

**. Physics Dept., Brandeis University, Waltham, Mass. 02154. This work was supported by the National Science Foundation under Grant Number DCR-74-24569.

BLANK PAGE

BLANK PAGE

# User Aids for MACSYMA[*]

by
V. Ellen Lewis
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts

## 1. SUMMARY

The aids available to the MACSYMA user are described, from the printed manual, primer, and system introduction to the various on-line sources of help. This is a tutorial paper which is, in fact, a "user aid" itself.

## 2. PRINTED MATERIAL

When a new user requests information about MACSYMA, he is sent a standard package consisting of the MACSYMA Reference Manual, the "MACSYMA Primer", and the "Introduction to ITS for the MACSYMA user." These three documents comprise the printed documentation for MACSYMA and are intended to provide enough information to a prospective user to permit him to (1) determine whether or not MACSYMA can help him solve his problem, and (2) get started using MACSYMA.

### 2.1. The MACSYMA Reference Manual

The Reference Manual is, of course, the most complete document dealing with the MACSYMA System. It describes all the functions, commands, switches and options available in the system. Most serious MACSYMA users will want to have one for reference. It has indices of functions and switches, as well as detailed information dealing with programming and the internal operation of MACSYMA. It is updated approximately every 12 to 18 months. In between

---

[*] This work was supported, in part, by the United States Energy Research and Development Administration under Contract Number E(11-1)-3070 and by the National Aeronautics and Space Administration under Grant NSG 1323.

revisions, information about new features is available on-line in the file `MACSYM;UPDATE >`[**].

### 2.2. The "Introduction to ITS" and the "MACSYMA Primer"

The "Introduction to ITS for the MACSYMA user" attempts to explain to those whose primary purpose in using the computer is using MACSYMA how to cope with the time-sharing system (ITS) on which MACSYMA runs. This is at best a stop-gap measure, but an essential one for the moment, because MACSYMA runs on a "research" system. The assumption is that the person using the computer wants to have access to any part of the operating system at all times. For a programmer this is a "feature" (an advantage), but for a user this can be a distinct disadvantage. The "Introduction to ITS" is intended to offset this disadvantage.

The MACSYMA Primer is a brief description of some of the commonly used features of MACSYMA. By use of a number of examples, it demonstrates MACSYMA's syntax and gives a short "cook book recipe" for how to use MACSYMA.

Using these two documents, a potential user can establish a connection to the computer, and get started using MACSYMA.

## 3. THE ON-LINE AIDS

MACSYMA is a system with a lot of built-in expertise. Once the user has gotten himself connected to it, it is reasonable to hope that MACSYMA can offer information about itself should the user desire it and respond to simple user queries.

### 3.1. The `PRIMER`

For the novice user, or other users who want some instruction in a particular aspect of MACSYMA, there is the on-line Primer. This is conceived as an interactive educational tool which leads the user through some sample calculations. It allows the user to type commands, but intercepts them for checking before they reach MACSYMA's evaluator. If a command is typed correctly, it is passed on to the evaluator and MACSYMA handles it exactly as if it had been typed in from top level MACSYMA. If a command is not correct, the Primer tries to identify the source of the error and give the user an appropriate error message. The command is not passed on to MACSYMA and the user is asked to "Try again." Thus the user gets "hands on" experience typing actual MACSYMA commands but in a controlled situation where he will be introduced to the complexities of the system without having to flounder around.

---

[**] This file may be printed out with the command `:PRINT MACSYM;UPDATE >`*<carriage return>* at DDT level

The command to start up the Primer is `PRIMER();`.[*] This will print out a brief introduction and offer a choice of subjects to learn about, thus:

```
(C1) PRIMER();

        Hello. Please terminate your responses with a ;. What would you
like to go over? (Select the number of the script you would like to see.)
INTRO is a general introduction for people who have never used MACSYMA or
this PRIMER before.
1 - INTRO
2 - SIMPLIFICATION
3 - SCRATCHPAD
4 - SYNTAX
5 - ASSIGNMENT
6 - FILING
7 - MATRICES
8 - SHARE
9 - EXIT
```

These topics are called scripts because their interactive nature makes them closer to dramatic scripts than to narratives. The user selects a script by typing its number (or its name) followed by a semicolon. (`INTRO` is the introductory script and should be run by new users.) There is a "standard" introduction consisting of the `INTRO` script (which inserts the `SYNTAX` script), the `SIMPLIFICATION` script, and the so-called `SCRATCHPAD` script[**]. These scripts lead one to the next, with an optional ordering offered. Additional scripts are available on `MATRICES`, `FILING` (the various kinds of disk files and how to use them), and `ASSIGNMENT` (how to define functions and assign variables). Scripts will eventually be added dealing with `EVALUATION`, program writing, and (in the spirit of self explanation) User Aids. Some of the information on the `SHARE` directory may also be printed out in the Primer, by selecting the `SHARE` script, which offers a further selection of file names to be printed. The `PRIMER` command may also be given a script name as an argument, e.g. `PRIMER(MATRICES);`, and it will then run that script.

The user is moved around from script to script in the Primer depending on how he answers the "yes or no" questions the Primer asks:

---

[*] This is called a "function of no arguments", since MACSYMA functions take their arguments inside parentheses.

[**] `SCRATCHPAD` is meant to imply the ability to "fiddle" with MACSYMA expressions. No connection with another manipulation system is intended.

```
          Do you need help with MACSYMA syntax?

     YES;
```

Other script switches are accomplished by the primer printing out the list of scripts again and allowing the user to select a script or to exit. (Also at any point the user may type control-uparrow, the MACSYMA "quit" character, and exit back to top level MACSYMA).

The user will be invited to try out the various commands as they are explained, e.g.

```
          Here is a simple example of the use of SUBST. The numerator of
     this expression is aqual to 1 for all X, but the MACSYMA simplifiers will
     not simplify it directly.

     (C2) (SIN(X)^2+COS(X)^2)/(X^2+39);
                                        2         2
                                     SIN (X) + COS (X)
     (D2)                            -----------------
                                           2
                                          X  + 39

          There are three ways to use SUBST on this example:
          One could substitute 1 for SIN(X)^2+COS(X)^2
          One could substitute 1-SIN(X)^2 for COS(X)^2
          Or one could substitute 1-COS(X)^2 for SIN(X)^2


          The first way is more direct, but in more complex examples where
     the sin squared plus cos squared is deeply entwined with other elements
     of the expression the second or third way would be necessary. Pick the
     way you like best and simplify the expression by using SUBST.
```

The user may then perform the indicated operation, or if he is not sure how to proceed (or has tried once or twice and been unsuccessful), he may type **NO;** and the Primer will show him how to do it:

```
     (C3) NO;

     O.K. I'll do it for you.
     (C3) SUBST(1,SIN(X)^2+COS(X)^2,%);
```

```
                                         1
(D3)                                  -------
                                        2
                                       X  + 39
```

### 3.2. The HELP Command

The casual MACSYMA user frequently wants to do one task, invert a matrix or solve a differential equation, for instance. The advanced user sometimes needs to know one thing like what switches affect a particular command. That is to say, there are specific questions users have, which fit into two general forms:

1. How do I *<do something>* ?
2. What are the *<arguments, switches>* for *<command>* ?

Of course, the user could ask a knowledgeable user these questions, or look them up in the Reference Manual, but this is not always convenient. So the `HELP();` command has been implemented. The `HELP();` command starts up a small "natural language" subsystem which can understand English in a flexible but limited way. Sentences it cannot understand are returned with the constructions or words the system does not undertand pointed out, so the user may rephrase his question. This HELPer is the beginning of the ADVISOR subsystem which will ultimately take the place of the communication with human advisors for most questions (see ref. 1).

Basically, this subsystem will be able to understand and reply to questions of the two forms stated above: "How do I _?" and "What are the _ for _?" The flexibility of the system permits, for instance, the two questions:

1. How do you append two lists?
2. How do I make one list out of two lists?

by recognizing that they are both requesting information about the APPEND command. Questions of the form

"How can I integrate D3?"

can also be handled, since the subsystem has access to the rest of the user's MACSYMA and can find out what D3 is, even replying "I'm sorry, MACSYMA cannot integrate <expression>," should that be the case.

To exit from the HELPer, type `BYE`.

## 3.3. Options, Describe, and Example

### 3.3.1 Options

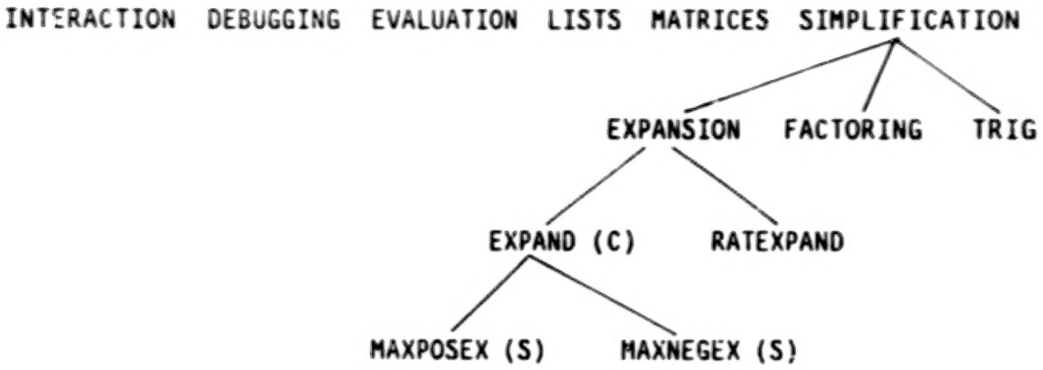
Users sometimes need to ask a more general sort of question, like "What can I do with a matrix?" or "What kinds of operations can I perform on trigonometric functions?" The `OPTIONS();` command was conceived for this purpose.

`OPTIONS();` starts up the "Options Interpreter". Note that `OPTIONS` may take the name of a command or a general topic (e.g. MATRICES, SIMPLIFICATION, FACTOR) as an argument. The effect of `OPTIONS();` is

```
(C4) OPTIONS();

OPTION FASL DSK MACSYM being loaded
loading done
OPTIONS interpreter  (Type "EXIT;" to exit.)
1 - INTERACTION
2 - DEBUGGING
3 - EVALUATION
4 - LISTS
5 - MATRICES
6 - SIMPLIFICATION
7 - REPRESENTATIONS
8 - PLOTTING
9 - TRANSLATION
```

This list of topics is the top of a branching hierarchical structure like an inverted tree which organizes the names of MACSYMA commands and switches by topic or function. A portion of the tree looks like this:


INTERACTION
DEBUGGING
EVALUATION
LISTS
MATRICES
SIMPLIFICATION
EXPANSION
FACTORING
TRIG
EXPAND (C)
RATEXPAND
MAXPOSEX (S)
MAXNEGEX (S)

The Options Interpreter uses the same mechanism for moving around in this tree that the Primer uses for script selection, thus referring back to the printout from **OPTIONS();**, the user types a number followed by a semi-colon to see the things under a particular topic (a "node" in the tree). For example:

```
(C4) OPTIONS();
OPTIONS interpreter  (Type "EXIT;" to exit.)
1 - INTERACTION
2 - DEBUGGING
3 - EVALUATION
4 - LISTS
5 - MATRICES
6 - SIMPLIFICATION
7 - REPRESENTATIONS
8 - PLOTTING
9 - TRANSLATION
6;
1 - EXPANSION
2 - FACTORING
3 - TRIG
```

Continuing further

```
1;
1 - EXPAND (C)
2 - RATEXPANU (C,S)
```

A command will have the symbol **(C)** after it, a switch will have the symbol **(S)**, and a variable will have **(V)**. Continuing down the tree, if the user selects "1", the **EXPAND** command, MACSYMA prints out:

```
1;
1 - MAXPOSEX (S)
2 - MAXNEGEX (S)
```

showing the switches which affect that command. If the user selects "1" at this point, the MAXPOSEX switch, MACSYMA prints out

```
          no options
```

indicating that he has reached the bottom of the tree. To move back up, perhaps to check out the RATEXPAND command, the user types

```
back;
1 - EXPAND (C)
2 - RATEXPAND (C,S)
```

and the system moves him back up to the next higher level. To exit from the OPTIONS Interpreter, type `exit;`.

### 3.3.2 Describe

The OPTIONS command allows the user to select a command or a small set of commands. The user can then check the command in the manual or use the DESCRIBE command to find out what it does exactly, and what arguments it takes. DESCRIBE takes a command name or a switch name as an argument and prints out the section of the manual which explains the command or switch.* DESCRIBE works within OPTIONS, taking the number of the command:

```
1 - FACTOR (C)
2 - GFACTOR (C)
3 - FACTORSUM (C)
4 - GFACTORSUM (C)
5 - SQFR (C)
6 - PARTITION (C)
DESCRIBE(1);
FACTOR(exp) factors the expression exp containing any number of
          variables or functions, into factors irreducible over
          the integers.
```

---

*. Of course, this is only as good as the latest version of the manual, and might be out of date if new features have been added.

Or DESCRIBE can be used directly from top level MACSYMA:

```
(C5) DESCRIBE(FACTOR);
FACTOR(exp) factors the expression exp containing any number of
        variables or functions, into factors irreducible over
        the integers.
(D5)                                DONE
```

### 3.3.3 Example

The EXAMPLE command fits very closely with DESCRIBE. It also takes a command as an argument and gives examples of how that command may be used, and the sort of output it gives.

```
(C6) EXAMPLE(FACTOR);

EXAMPL 2 DSK DEMO being loaded
loading done

(C7) FACTOR&& FACTOR(2^63-1);

                          2
(D7)                     7  73 127 337 92737 649657

_
(C8) FACTOR(Z^2*(X+2*Y)-4*X-8*Y);
(D8)                     (2 Y + X) (Z - 2) (Z + 2)

_
```

Since the EXAMPLE command is actually a demonstration (see DEMO command below), it prompts the user with a "_" at the left margin after each command line is processed, so the user may type a space to see the next command line, or control-uparrow to "QUIT" out of the EXAMPLE.

## 3.4. Demonstrations, and the DEMO Directory

Another way a user can find out how various MACSYMA functions work and get an idea of how MACSYMA can be used on real problems is to run some of the demonstrations which are contained in the DEMO directory.

The directory may be listed at system top level (DDT level)[*] and the files loaded into MACSYMA with the DEMO command, e.g.

```
(C9) DEMO(NDEMO,FILE,DSK,DEMO);
```

## 4. USER SPECIFIC INFORMATION

All the user aids discussed thus far have been for getting information about the system. It is sometimes necessary for a user to get information about his own functions or the current state of his MACSYMA.

### 4.1. Information about User-Defined Functions and Variables

### 4.1.1 DISPFUN and GRIND

Suppose the user has defined a function F(X), for instance:

```
(C10) F(X):=X^2+2*X+1;
                                             2
(D10)                          F(X) := X  + 2 X + 1
```

The user can redisplay this function using the command DISPFUN(F);

```
(C11) DISPFUN(F);
                                             2
(D11)                          F(X) := X  + 2 X + 1
```

In this way he can check the correctness of the definition, or review it.

If the function the user had defined is a BLOCK statement, e.g.

```
(C12) MYTAYLOR(EXPR,VAR,POINT,HIPOWER):=BLOCK([RESULT],
RESULT: SUBST(POINT,VAR,EXPR),FOR I:1 THRU HIPOWER
DO (EXPR: DIFF(EXPR,VAR)/I,RESULT: RESULT+(VAR-POINT)^I*
SUBST(POINT,VAR,EXPR)), RETURN(RESULT))$
```

---

[*] :LISTF DEMO<*carriage return*>

just displaying it may not be very helpful, especially if the user is trying to "debug" it. The command GRIND(G); can be used and will display the function G with the various parts of the **BLOCK** statement indented properly so their structure can be more easily seen, for example:

```
(C13) GRIND(MYTAYLOR);
MYTAYLOR(EXPR,VAR,POINT,HIPOWER):=BLOCK([RESULT],
          RESULT:SUBST(POINT,VAR,EXPR),
          FOR I THRU HIPOWER DO
              (EXPR:DIFF(EXPR,VAR,1)/I,
               RESULT:RESULT+(VAR-POINT)^I*SUBST(POINT,VAR,EXPR)),
          RETURN(RESULT))$
(D13)                             DONE
```

Using GRIND on a function like F(X) above (which fits on one line) produces the one dimensional representation in which the function was input, although in general it might be equivalent but slightly re-arranged.

```
(C14) GRIND(F);
F(X):=X^2+2*X+1$
(D14)                             DONE
```

### 4.1.2 PROPERTIES and ARRAYINFO

The command PROPERTIES takes a function or a variable as an argument, and prints out the things MACSYMA knows about it, e.g. that it is a function. For example:

```
(C15) PROPERTIES(MYTAYLOR);

PROPFN FASL DSK MAXOUT being loaded
loading done
(D15)                         [FUNCTION]

(C16) PROPERTIES(GRIND);
(D16)                     [SYSTEM FUNCTION]
```

The command ARRAYINFO takes the name of an array as an argument, and will print out the information about the array: whether or not it is declared and its dimensions.

## 4.2. INFOLISTS

INFOLISTS is a list of the lists of information MACSYMA maintains about the user's MACSYMA state. Typing INFOLISTS; will produce the following output:

```
(C17) INFOLISTS;
(D17) [LABELS, VALUES, FUNCTIONS, ARRAYS, MYOPTIONS, PROPS, ALIASES,
                                 RULES, GRADEFS, DEPENDENCIES, FEATURES]
```

EV(INFOLISTS); will produce a list of the things in each of the lists. The lists maintained are:

**LABELS** - The line labels in the current MACSYMA which have been assigned, that is all C-lines, D-lines, and E-lines.

**VALUES** - All the variables the user has assigned a value to explicitly with the : operator, by variable name.

**FUNCTIONS** - All the functions the user has defined with the := operator, except subscripted (array) functions.

**ARRAYS** - All arrays and matrices, declared and undeclared, and all array functions.

**MYOPTIONS** - All the MACSYMA options (switches) the user has changed.

**PROPS** - Any atoms which have properties such as atvalues, matchdeclares, or properties specified by the DECLARE function.

**ALIASES** - The user's own abbreviated names for quantities, e.g. ALIAS(INTEG,INTEGRATE) sets up INTEG as a short spelling for INTEGRATE.

**RULES** - Any simplification rules or pattern matching rules the user has defined using the TELLSIMP, TELLSIMPAFTER, DEFMATCH, or DEFRULE commands.

**GRADEFS** - Those functions for which the user has defined derivatives.

**DEPENDENCIES** - The functional dependencies declared by the user with the DEPENDENCIES or GRADEF command.

**FEATURES** - Special mathematical or other properties of functions. Three are built into MACSYMA: INTEGER, EVEN, and ODD, but the user can add others.

## 4.3. Tracing and Debugging Aids

The TRACE function accepts the names of functions as arguments, and will print out information each time the functions being traced are called, e.g.

```
(C18) TRACE(MYTAYLOR);

MTRACE FASL DSK MACSYM being loaded
loading done
(D18)                          [MYTAYLOR]

(C19) MYTAYLOR(SIN(X),X,A,3);
1 enter MYTAYLOR [SIN(X), X, A, 3]
                                        3                  2
                         COS(A) (X - A)    SIN(A) (X - A)
1 exit MYTAYLOR: - --------------- - --------------- + COS(A) (X - A)
                               6                  2

                                                                + SIN(A)
                      3                  2
         COS(A) (X - A)    SIN(A) (X - A)
(D19) - --------------- - --------------- + COS(A) (X - A) + SIN(A)
                 6                  2
```

This permits the user to make a better guess as to where his function is not behaving as he expects.

The UNTRACE function is the complementary function which removes the trace from functions (e.g., UNTRACE(MYTAYLOR);). UNTRACE(); will remove tracing from all functions. TRACE(); will print out a list of all functions being traced.

```
(C20) TRACE();
(D20)                          [MYTAYLOR]

(C21) UNTRACE();
(D21)                          [MYTAYLOR]
```

There is a switch which helps the user keep track of what variables he has assigned values to. This is SETCHECK. SETCHECK may be set (using the : operator) to a list of variables, and **MACSYMA** will print out a message any time an assignment is made to one of those variables.

There are a few other debugging aids, which are explained in the manual in the section on Debugging Functions.

## 5. FINALLY, THERE ARE STILL PEOPLE!

Finally, should the user find these various aids inadequate, there are still human advisors around to whom he can put his questions. These human advisors are MACSYMA's best "User Aid", and the user is encouraged to contact them with his problems. This can be done within MACSYMA by using the SEND command, e.g.

```
(C22) SEND("HOW DO I INVERT A MATRIX?");
```

Notice the quotation marks, they are part of the command. This will send a message to one of the MACSYMA helpers who is logged in at the time. Alternatively, the user desiring help can exit from MACSYMA with a control-Z and use the DDT command :SEND to contact a particular person[°], or in cases of desperation, the :LUSER command.[°°]

## 6. REFERENCES

1. Genesereth, M. R.: "An Automated Consultant for MACSYMA". 1977 MACSYMA User's Conference, NASA CP-2012, 1977 (paper no. 30) of this compilation.

---

[°] See the "Introduction to ITS for MACSYMA Users" for details

[°°] Once again, see the "Introduction to ITS."

# The Difficulties of Using MACSYMA and the Function of User Aids[*]

Michael R. Genesereth

Center for Research in Computing Technology
Harvard University

Laboratory for Computer Science
Massachusetts Institute of Technology

## Abstract

The difficulties of using a computer system to help solve a problem can be divided into learning difficulties, resource knowledge difficulties, and communication difficulties. The purpose of this paper is to explore the nature and manifestations of these difficulties in MACSYMA and to explain the function of user aids in dealing with them. A *learning difficulty* arises whenever a system is too large or too complex to understand fully. A *resource knowledge difficulty* arises whenever a user is unable to solve his problem due to a deficiency in this understanding. A *communication difficulty* is due to a difference between the primitive objects, actions, and relations of a user's problem and those provided by the system. The importance of this distinction lies in the way each difficulty is handled: learning difficulties by primers, lectures, tutors; resource knowledge difficulties by manuals, information networks, consultants; communication difficulties by bringing the system closer to the user's needs. In all cases, the optimal assistance can be provided by an aid that maintains and uses an explicit, internal "model" of the user's state of knowledge, his goals, and his "plan" for achieving them.

## Introduction

Consider a scientist trying to solve a mathematical problem with the aid of an algebraic manipulation system like MACSYMA. If he were to solve the problem by hand, he would personally have to grapple with the problem itself and all the subproblems that arise. By using MACSYMA, he can delegate many subproblems and thereby save time and effort. However, to do so, he must (1) understand the relevant portions of MACSYMA, (2) be able to remedy any difficulties that arise from a deficiency in this understanding, and (3) expend the additional effort necessary to communicate to MACSYMA the essential details of his problem. In general, when a person employs any tool to help solve a problem, he is trading off the effort required for these three tasks in return for the tool's powerful or unique abilities at solving his problem.

---

[*] This work was supported, in part, by the United States Energy Research and Development Administration under Contract Number E(11-1)-3070 and by the National Aeronautics and Space Administration under Grant NSG 1323.

The purpose of this paper is to explore the nature and manifestations of these tasks in the context of MACSYMA and to explain the function of user aids in facilitating their execution. People sometimes complain that MACSYMA is difficult to understand or to control, and they usually cite specific properties of the system as primarily responsible, e.g. too many commands, too hard to specify subexpressions. In all cases, these complaints are attributable to increases in the difficulty of one or more of the above tasks. Difficulties encountered in acquiring an initial understanding of a system will hereafter be called learning difficulties; problem solving difficulties resulting from a deficiency in this understanding will be called resource knowledge difficulties; and difficulties in communicating to the system the essential details of a problem and in retrieving a comprehensible result will be called communication difficulties. The importance of this distinction lies in the way each difficulty can best be handled. All three difficulties can be lessened by improving MACSYMA itself. However, learning difficulties can also be treated by tutorial aids, and resource knowledge difficulties by "user-initiative" information sources. It will be argued that in all three cases the ultimate aid is one that maintains and uses a "model" of the user's problem and his "plan" for solving it.

The analysis presented here is concerned only with difficulties arising from the use of MACSYMA; it does not consider those arising from ill formulated or partially formulated problems. Such problems are not uncommon, e.g. a scientist will occasionally engage in algebraic manipulation without a precise goal because he wants the insight that comes from writing his result in different forms. Although the paper does mention in general terms the constraints on MACSYMA's design, it does not consider specific implementational or mathematical difficulties, e.g. address space problems, the representation of derivatives.

A learning difficulty arises when a system is too large or its primitives too complex for a new user to understand fully. MACSYMA, for example, has over 350 commands and 200 switches, and the behavior of many commands like TRIGREDUCE cannot be simply described. Learning difficulties are best countered either by simplifying the system or by providing tutorial aids like primers and lectures.

A resource knowledge difficulty arises when the user finds himself unable to proceed further in solving his problem due to a deficiency in his knowledge of MACSYMA. He might not, for example, be able to remember the name of the command for putting a sum of quotients over a common denominator (COMBINE). Or, he might be unaware of a command's dependence on the setting of some variable, e.g. EXPAND and MAXPOSEX. Or, he might get an incorrect answer due to a programming mistake but not know where in his derivation he went wrong. Resource knowledge difficulties are best treated by user-initiative information sources, e.g. manuals, information networks, and consultants.

A communication difficulty results from a difference between the objects, actions, and relations of the user's problem and those provided by the system. The difference may be either simple or complex. A "simple" difference is eliminated by defining the relevant concepts. For example, MACSYMA can represent a matrix and compute and solve its characteristic polynomial, but it knows nothing about eigenvalues. The user with a matrix eigenvalue problem may either call the appropriate commands one by one or define a function. A "complex" difference results when there is no homomorphic mapping between the primitives of the user's problem and their representation in MACSYMA. For example, a user may want to write an expression as $(V/C)^2$, but MACSYMA insists on writing $V^2/C^2$. The most straightforward solution to communication difficulties is for the system designer to bring the system's primitives closer to those of the user.

It is important to keep in mind a basic distinction between learning and resource knowledge difficulties on the one hand and communication difficulties on the other. A communication difficulty results from the difference between the expertise required to solve the user's problem and that provided by the system. A learning or resource knowledge difficulty is due to the user's misunderstanding of the system, no matter how appropriate the system is to the problem at hand. A communication difficulty varies inversely with the system's expertise and would exist even if the user understood MACSYMA perfectly. Learning and resource knowledge difficulties vary directly with the complexity of that portion of the system appropriate to the user's problem and are otherwise independent of the problem.

The advantage of a large algebraic manipulation system like MACSYMA over a smaller, sparer system like REDUCE is that MACSYMA has more mathematical knowledge built in. As a consequence, the user is not forced to communicate as much mathematical knowledge to the system, and it is even possible that the system offers expertise with which the user himself is unfamiliar. The disadvantage is that MACSYMA can be more difficult to understand and to use. In other words, the communication difficulty is drastically decreased for increased learning and resource knowledge difficulties.

One advantage of numerical computation over symbolic manipulation is that the former can sometimes succeed where the latter fails -- many problems are amenable only to numerical techniques. This is unfortunate because graphs and tables alone do not offer as much structure as closed form or even series solutions. The inadequacy of numerical solutions can be viewed as a communication difficulty in which the answers are not as readily interpretable in the user's terms. Thus, when both numerical computation and symbolic manipulation are applicable, the latter has the advantage of more comprehensible results and, due to the decreased communication difficulty, may actually be more efficient in terms of user time.

In providing the optimal assistance for each of these three types of difficulties, one feature is common, namely the importance of a model for the user's goal and his plan for achieving it. In order to provide information tailored to the user's need, the tutor or consultant must know what the user knows and what he is trying to do. If MACSYMA were able to keep track of the structure of the user's session (why he is doing what he is doing), it could choose defaults and disambiguate input in a way that is not now possible. The automatic user aids of the future -- tutors, consultants, and apprentices -- will very likely maintain and use such models.

This paper deals with the three types of difficulties in turn. The first section describes MACSYMA's tutorial aids, discusses their strengths, and suggests some improvements. The second section classifies and explains the observed manifestations of resource knowledge difficulties by way of an explicit model of the "typical" MACSYMA user and describes MACSYMA's provisions for dealing with these difficulties. After listing the requirements for communication with MACSYMA, the third section outlines its current capabilities for easing communication difficulties and suggests several improvements that would further reduce their degree. The fourth section states in very general terms v[illegible] MACSYMA has developed as it has. The final section describes the state of implementation [illegible] suggestions made in the paper, indicates some shortcomings of the model used in section 2, and argues that the difficulties of using a computer system need not be prohibitive if adequate user aids are provided.

## 1. Learning Difficulties

A learning difficulty arises when a system is too large or its primitives too complex for a new user to understand fully. The effect of having too many commands and switches is that the user cannot remember all the capabilities available and the details of each; there is just too much information. A mnemonic naming scheme is one way MACSYMA tries to counter this difficulty. Obviously, a good naming scheme should be unambiguous, systematic, prescriptive, and designative of the command's exact function.

Mnemonic naming is the best way to help a user recall the name of a command or switch. However, the best way to help him remember the range of capabilities available is to provide a conceptual framework for those capabilities. A primer is a user aid that supplies information from a fixed syllabus. This facilitates the learning process by structuring the material to be learned. MACSYMA has a small hard copy primer (ref. 1) that is supplied to all new users.

The best way to help a user remember the details of a command's use, e.g. its arguments, options, side effects, is practice. MACSYMA also has an interactive primer (ref. 2) in which the user participates by solving test problems under its auspices (via the PRIMER command). The advantage is that the user is forced to try out what he has learned immediately after he learns it. The user's solution is checked for mistakes by specialized analysis functions supplied by the primer's author.

In the future, this analysis and maybe even the invention of examples may be automated. The work reported in (refs. 3, 4, 5, 6) suggests a possible implementation. The MACSYMA tutor would maintain a model of the user's knowledge of MACSYMA based on the material already presented to him and a model of the task he was given; and it would obtain through analysis of his actions and statements a model of his plan for solving the problem. It would examine these models in an attempt to recognize any tutorial "issues" (ref. 4) in its syllabus and, finding one, would generate the appropriate correction. The construction of such a tutor, however, has not yet been seriously considered.

One other tutorial approach is the traditional lecture and problem set discipline. The MACSYMA staff yearly offers a six lecture mini-course at M.I.T., and there are plans to videotape these lectures for general circulation.

The disadvantage of a tutorial aid is that the information provided is not tailored to the user's current problem. While a full presentation may be best in the long run, some users may not have the time or patience to consult such an aid before tackling their problem.

## 2. Resource Knowledge Problems

The MACSYMA user typically has a mathematical problem he is trying to solve and approaches MACSYMA for its powerful abilities at algebraic manipulation. The domain in which the problem is expressed (here mathematics) is called the task environment, and the user typically knows a good deal about it. This knowledge is represented in figure 1 as the box labeled **T**. He also has a model of MACSYMA's abilities (**M**) and maintains a dynamic model for the state of his current MACSYMA (m). In solving his problem, the person uses this knowledge to map his problem from the task environment to MACSYMA, solve the resulting MACSYMA problem, and interpret the result. For example, he represents his equations as a matrix, inverts it, and reads off the solutions. In executing this procedure, he implicitly generates and follows a plan **P**, i.e. a goal-subgoal tree that he believes will solve his problem. This view of the user's use of MACSYMA leads to the configuration in figure 1.

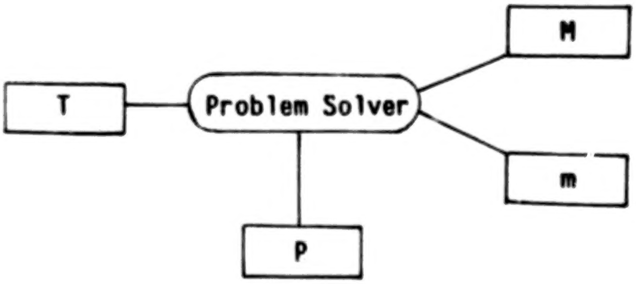


Fig. 1 - A MACSYMA user's data structures

A resource knowledge difficulty arises when a user is unable to proceed further in solving his problem due to a deficiency in his model of MACSYMA (**M**). When this happens, the user must either strike out at random or consult one of the information sources available to him. Difficulties due to errors in the user's model of his task environment (**T**) are not treated here, though they often arise. One might, for example, balk at seeing an imaginary solution when trying to find the intersection of two circles, until one realizes that the circles do not intersect. Difficulties due to deficiencies in the user's model of his current MACSYMA (**m**) stem from deficiencies in **M** or **T** and are dealt with in part by improving communication of MACSYMA's state to the user as described in section 3.

In analyzing resource knowledge difficulties, several questions naturally arise. Is there any way to bound and classify the sorts of difficulties that can befall the user? Of what use are user aids in dealing with these difficulties? This section presents some data on the information needs of users experiencing resource knowledge difficulties and explains this data by way of a model of the "typical" MACSYMA user.

## 2.1 Observed Information Needs of MACSYMA Users

One of MACSYMA's strongest user aids is its staff of human consultants, available on-line to help users with resource knowledge difficulties. During the last three years, the author has served as a MACSYMA consultant and recorded many of these consultation sessions. During the same three years, Profs. Gorry, Martin, and Szolovitz have offered a course on "knowledge-based systems" at M.I.T. in which one of the requirements is the solution of a MACSYMA problem and an analysis of the resulting protocol. The analyses were supposed to indicate which information sources were consulted and why. The author also had the opportunity to read many of these analyses.

An examination of the data obtained from such consultations and protocol analyses reveals that in using MACSYMA, people perceive the need for five general classes of information.

(1) The user needs to know the name of a command or technique to do some task. If he were to phrase his need as a question, he would ask "How do I do ..?" This is called a HOWDO need.

(2) He needs to know a command's prerequisites, arguments, postrequisites, etc. He would ask "What are the ... of ...?". A WHAT need.

(3) He needs to check his beliefs about MACSYMA. He would ask "Is it the case that ...?". An IS need.

(4) He needs a procedural explanation of how a command works or a result was obtained. He would ask "How did MACSYMA do ...?". A HOW need.

(5) MACSYMA has returned an unexpected result, and he can find nothing wrong with his derivation. He needs sufficient information to pinpoint and correct the misconception underlying his erroneous expectation. He would ask "Why is it that ...?" A WHY need.

Of course, the syntax the person uses need not correspond to these five categories, only the underlying question. For example, "Can you tell me how to invert a matrix?" means "How do I invert a matrix?" and a complaint of "D13 is positive!" means "Why is D13 positive?".

## 2.2 A Model for the "Typical" MACSYMA User

The analysis presented here assumes that in solving his problem the user acts in accordance with a standard, high level planning algorithm. This algorithm is best represented as a "state and transition augmented network" (called SATAN) in which the states represent problem solving commitments and the transitions are augmented by predicates and problem solving actions (accesses and updates to M, m, and P). For the present discussion, however, the full network described in (ref. 6) may be simplified to the flowchart in figure 2.

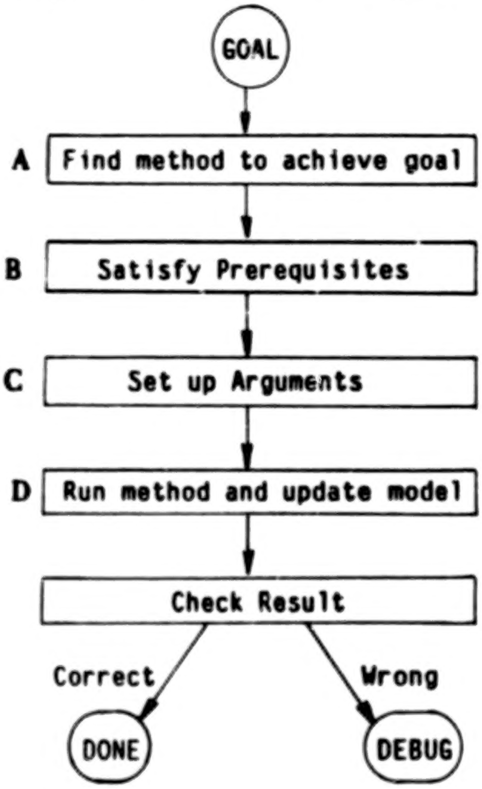


Fig. 2 - A flowchart for the "typical" user's planning strategy

The initial goal is the solution of the MACSYMA version of the user's problem. In processing a goal, the problem solver either selects a "canned" method (a "template") or develops one especially from the facts about the objects and relations involved. The method chosen may be a single command or a high level program with commands and other goals as steps (a "procedural net"). In processing these subgoals, the problem solver generates yet other procedural nets until a level is reached containing only MACSYMA commands. Thus, the normal operation of the problem solver implicitly generates a hierarchical goal-subgoal tree, the root of which is the user's ultimate goal and the fringe of which is his MACSYMA solution. At any given level, the problem solver may insert additional goals to achieve prerequisites or check results. It may also transform the plan, omitting or rearranging steps, in order to optimize it. This step is not shown in figure 2. This means that the goal "tree" may in fact become a directed acyclic graph. It is important to remember that the plan need not be explicit, i.e. the user need not be conscious of his plan; the essential point is that the user acts as if he were following a plan.

During the planning process, the user forms expectations about the results of his plan. When he checks these results, however, he may discover a discrepancy between these expectations and the facts (a bug manifestation). This discrepancy may be due either to a simple planning or execution mistake, e.g. a sign error, or to a more significant deficiency in **M**, **m**, or **T** (called a misconception). However, the point in his plan at which the misconception had its effect (the locus) may not be immediately apparent. If so, the user must pinpoint the locus in order to uncover the misconception. In debugging his plan, the user is assumed to operate in accordance with a standard, high level debugging algorithm. Like the planning algorithm, this algorithm is also best described as an augmented network. However, for the present purposes, it can be simplified to the flow chart in figure 3.

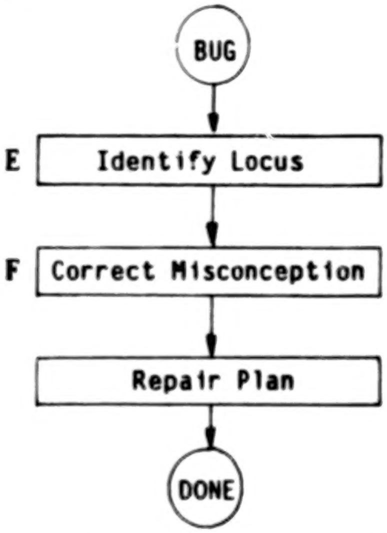


Fig. 3 - A flowchart for the "typical" user's debugging strategy

It is when the user finds himself unable to perform any of the steps in the planning or debugging procedures due to a lack of knowledge about MACSYMA (deficient **M**) that a resource knowledge difficulty becomes manifest.

(1) A HOWDO need arises in box **A** of the planning algorithm.

(2) WHAT needs arise in boxes **B**, **C**, **D**.

(3) An IS need can arise in any box, but most often in debugging.

(4) The user may be unable to identify the locus of a misconception in box **E** of the debugging algorithm and therefore experiences a HOW need.

(5) He may be unable to find anything wrong with his plan, i.e. he needs help in either box **E** or **F**. This is a WHY need.

According to this mechanistic model of MACSYMA problem solving, a resource knowledge difficulty is viewed as the user's inability to make a transition from some problem solving state, and the kind of difficulty that arises identifies the offending state. The importance of having such a model is that it explains how resource knowledge difficulties arise and sets a neatly specifiable bound on the types of difficulties and thereby on the types of assistance that user aids must provide.

## 2.3 The Function of User Aids

In order to deal with the difficulties listed in section 2.1, system designers often provide an array of user aids.

The most common aid is the system's reference manual. MACSYMA's manual is available both in hard copy and on line (via the DESCRIBE command). The function of a manual is to provide quick reference to the facts about a command or variable, given its name. Thus, a manual effectively satisfies WHAT needs and many IS needs.

Also common is the system's trace capability. MACSYMA allows a user to trace all function entries and exits (the TRACE command) as well as the settings of variables (the SETCHECK variable). The purpose of tracing is to help the user discover the locus of the misconception underlying his bug manifestation, and therefore it helps meet HOW needs.

A less common user aid is the "inverted manual", or information network. MACSYMA's version of this is available via the OPTIONS command. An information network is essentially a thesaurus of commands indexed by category and is primarily intended to help the user find the commands applicable to a particular task. Its primary effect is to answer HOWDO questions.

WHAT, HOWDO, and IS problems can be dealt with directly by an information source with no sensitivity to the user's purposes or state of knowledge. A WHY or HOW problem, however, often calls for different answers to different people in different situations. Such a problem arises when a misconception gives rise to a bug manifestation, and its treatment calls for providing the user with enough information to correct the misconception. A source able to provide just this information and no more must have a model of the user's state of knowledge (**M**), his model of the current MACSYMA (**m**), his goal (**T**), and his plan for achieving it (**P**), and it therefore must be considerably more sophisticated than the other, user-independent aids. A consultant is an information source that seeks to improve the user's model of the system in "user-initiative" mode. Consultation is a method widely used in computer centers for coping with WHY and HOW questions, and MACSYMA's consulting staff has proved to be its most effective user aid. A consultant can deal with all five kinds of problems and provide information tailored to the user's need and level of understanding. Armed with the consultant's advice, the user can often surmount his difficulty and continue solving his problem.

Unfortunately, human consultants are a scarce resource and quite expensive. And, as MACSYMA is exported and its user community grows, even more consultants might have to be provided. For this reason, work has begun on the construction of an automated consultant, called the Advisor. This program should be able to converse with the user in English about a difficulty he has encountered and provide advice tailored to his need. The MACSYMA Advisor is a program distinct from MACSYMA with its own separate data base and expertise. However, for convenience the program can be called directly from MACSYMA (via the HELP command) and can access the user's data structures. As currently implemented, the Advisor deals only with the "straight line" or nested use of MACSYMA commands and not loops or user-defined functions. For a concrete example of the Advisor's performance, one should see the abstract printed in these proceedings. As with the proposed MACSYMA tutor, the MACSYMA advisor relies heavily on its partial models of the user's state of knowledge, his goal, and his plan for achieving it.

## 3. Communication Difficulties

A communication difficulty is the result of the difference between the primitive objects, actions, and relations in the user's problem and those provided by the system. Thus, the degree of such a difficulty is a function of both the user's problem and the system's expertise. Although a resource knowledge difficulty can be thought of as a communication difficulty, the concern here is with those difficulties that remain even when the user's model of MACSYMA is complete.

The difference may be either simple or complex. A simple difference is eliminated by defining the relevant concepts. For example, MACSYMA can represent a matrix and compute and solve its characteristic polynomial, but it knows nothing about eigenvalues. However, the user with a matrix eigenvalue problem may educate MACSYMA simply by defining a function that calls the appropriate commands. The disadvantage of a "conservative" system (ref. 7) is that the user must convey large amounts of knowledge in this form. A complex difference results when there is no homomorphic mapping between the primitives of the user's problem and their representation in MACSYMA. For example, a user may want to write an expression as $(V/C)^2$, but MACSYMA insists on writing $V^2/C^2$. Or, a user may define his operators by the identities they satisfy, but MACSYMA insists on function definitions and unidirectional replacement rules. The disadvantage of a "radical" system (ref. 7) is that its "model" of algebraic manipulation is in some domains too narrow and rigid to accommodate the full range of models possessed by users. Some recent work on reformulating problem descriptions expressed in the user's language in terms of a system's model of the domain is reported in (ref. 8). However, no such capability is yet available in MACSYMA, and so the user must translate his problems into MACSYMA's terms. Fortunately, MACSYMA is, within limits, a diverse system offering both radical representations where applicable and a flexible general representation otherwise.

The communication task consists of breaching the distance between the user's problem and the appropriate system model. The necessary information that must be conveyed to the system includes:

(1) input expressions, constraints, and domain-dependent expertise, e.g. inequalities, order truncation information, physical arguments

(2) operations to be performed, e.g. solving an equation, showing two expressions equal

In evaluating the degree of the input communication difficulty, the two most important issues are the amount of material that must be presented and the degree of flexibility in order and format of its presentation. The information that must be retrieved from MACSYMA includes:

(3) form of the solution, e.g. "expanded in Z"

(4) information about MACSYMA's state, e.g. values of switches

Furthermore, the user might want an explanation of how the result was obtained. If the system's model is similar to the user's, the explanation should be quite simple, e.g. integration by parts; if the technique used is very different, the explanation might be more complicated, e.g. explaining the whole Risch algorithm. Recent work reported in (refs. 6, 9, 10) indicates how a system could be made to explain its behavior.

## 3.1 Present Capabilities in MACSYMA for Facilitating Communication

Occasionally, a user may want to update or verify his model of the current MACSYMA (m). For this purpose, MACSYMA has a full set of information commands and variables. These differ from the commands mentioned in section 2 in that they provide information about the state of the user's particular MACSYMA and not about MACSYMA in general. These sources fall into two categories: finding information about an object given its name, e.g. DISPFUN, DISPRULE, and PRINTPROPS, and finding the names of all objects having a given feature, e.g. VALUES, FUNCTIONS, GRADEFS, etc. The sources available are listed in (ref. 2) and described in detail in (ref. 11).

Very often MACSYMA produces large, unwieldy results affording little insight. In a recent paper (ref. 12), David Stoutemyer discusses a package written in MACSYMA to extract the "qualitative" features of an expression, e.g. its sign behavior, convexity or concavity, zeros, periodicity, etc. For users as interested in the qualitative behavior of an expression as its symbolic details, this package should be of great value. It attacks the communication problem through item (3) of the above list.

The idea behind a specialized "application package" is to convert MACSYMA into an expert in a given domain and thereby lessen communication difficulties. A good example is MACSYMA's explicit tensor manipulation package. Another example is the forthcoming TRANSLATE helper that will lead the user by the hand through the translation and compilation process. The tensor package brings with it much knowledge that the user would otherwise have to communicate himself. The TRANSLATE helper guides the user's activities according to a model of the translation process and thereby saves problem solving effort on the part of the user. In this manner, these packages convert MACSYMA in limited domains from its normal "operator-based" mode into a "model-based" system.

## 3.2 Suggested Improvements to MACSYMA

When a person chooses to employ any tool to help him solve a problem, even if he has a complete model of how it works, he must expend the effort necessary to specialize the tool (e.g. define functions in MACSYMA, build jigs for a woodworking machine), and transform his problem into an amenable form, (e.g. represent his linear equations as a matrix). Obviously, some tools are better suited to a given problem than others. Among computer systems, two extremes stand out, namely the expert problem solver and the programming language.

An expert is an agent with language, knowledge, and abilities tailored to a particular domain and able to solve any reasonable, appropriate problem without outside guidance, e.g. an electronic circuit analysis program like SCEPTRE. Assuming the expert is flexible about input and does not employ too alien a model, the user need only describe his problem, then sit back and wait for the answer. Communication difficulties, among others, are minimal. In fact, item (4) above is completely unnecessary.

The approach of programming language designers is to provide some computational primitives useful to the user in writing code to solve his problem. Usually, the user must contribute his own problem solving skills in writing the code. The meaning of a primitive is usually independent of the use to which it is put, e.g. COEFF works the same whether the problem is solving a quadratic or computing syzygies. The lower the "level" of the primitives, the greater the simple differences between the user's world and the system's but the fewer the complex differences.

MACSYMA is primarily a programming language, albeit a very high level one, with only a few expert question-asking submodules, e.g. the tensor package. One could imagine, though, a system somewhere between these two extremes. It would keep track of the user's goals and actions and terminology and would use this information to facilitate input and try to solve his problem using a mechanical problem solver able to take advice from the user at crucial points. This possibility is discussed further below.

Several of the ideas presented in this section are concerned with the conception of mathematical knowledge as a body of programming rules, implemented in MACSYMA as variable values, function definitions, TELLSIMP rules, etc., rather than as a set of mathematical definitions and constraints. A rule in MACSYMA consists of (1) an identity and (2) an application procedure. An identity is always interpreted as a unidirectional replacement rule, i.e. whenever an expression matches the left hand side of an identity, it is replaced by the right hand side and never the other way around. The match procedure is for the most part "local". Although global conditions can be tested in the predicates constraining the variables of a TELLSIMP rule, the properties of the expression enclosing the one being matched cannot be easily checked. And, most significantly, there is no sensitivity to the user's goal or plan, no overall direction to decide when a replacement rule should be made and when bypassed in order, for example, to achieve a cancellation or prevent an infinite loop.

There are various types of application procedures. Some rules are applied at only a single level, e.g. XTHRU, MULTTHRU. Others have automatic recursion built in, e.g. TRIGEXPAND, LOGCONTRACT, TELLSIMP rules. The application is in all cases deterministic, despite the possibility of a non-unique match between the pattern and the expression, e.g. matching X+Y to A+B+3.

When a user complains that MACSYMA is too hard to control, he is usually referring to its lack of selectivity in the automatic, recursive application of evaluation or simplification rules. MACSYMA provides automatic recursive application to save the user the drudgery of applying a large body of system-defined and user-defined rules by hand. However, the user may occasionally want a rule to be applied nonuniformly, e.g. when evaluating only certain derivatives in an expression after plugging in values for some variables. Or, he may want a rule applied in reverse. Due to MACSYMA's unidirectionality, this requires that a second rule be defined, which can result in an infinite loop. In using automatic, recursive rule application, the user is sacrificing the effort necessary to control MACSYMA to eliminate the drudgery of applying the rules himself.

In order to avoid the complications that can arise from the user's ignorance of the rules used by the general simplifier and commands like INTEGRATE, these rules should be made explicit and controllable. This suggestion has already been realized in the realm of trigonometric simplification, where all rules are named and can be activated or deactivated by the setting of a switch. It would be convenient if the "," syntax at top level MACSYMA could be extended to activate rules for one line's duration just as it is now used to define substitution rules. With this syntax one would be able to say, for example, D4,X=2,$Z^2$=4,SINRULE1,EXPONENTIALIZE. This suggestion is in keeping with the view of the "," syntax as an "environment setup" command.

More generally what is needed is a better structuring of simplification rules. It is doubtful that a user would define rules for the internal use of heuristic commands since their operation usually is too complex to describe. Therefore, complex commands like INTEGRATE should deactivate all potentially conflicting user rules until their work is done. One way of implementing this that would offer other desirable features is in the form of "environments": sets of rules, variable bindings, function definitions, declarations, and assumptions that can be "shallow bound". A primitive form of environment structuring is already available in MACSYMA through the context mechanism. As with contexts, environments should be hierarchically structured. It would then be possible for the environments for certain domains, like gravitation heory and continuum mechanics, to share the knowledge of common subdomains like tensor manipulations, while remaining distinct from conflicting domains like Newtonian physics.

Another improvement would be the ability to add properties to expressions as well as variables. It is currently possible to declare partial information about variables, e.g. DECLARE(N,INTEGER); however one cannot declare similar information about expressions even though it might be useful for later manipulations. For example, in integrating an expression, the user might make an assumption about the sign of a variable that could be used by he LIMIT command at a later time. The new MACSYMA internal representation together with

MACSYMA's high level data base system (ref. 13) should be able to represent such information quite easily. Furthermore, it should allow the user to tell MACSYMA the semantic significance of expressions, e.g. that G*V/M is a convection term, and to define semantic rules to prevent combining semantically incompatible terms, e.g. adding apples and oranges. This ability is available now only in the restrictive form of the "invisible boxes" generated by the TBOX command.

Perhaps the most ambitious suggestion is to transform MACSYMA from the programming language that it is now into a more intelligent, problem solving system, a sort of "mathematician's apprentice". The essential idea behind this proposal is for the system to maintain and use information about the user's goal and his plan for achieving it. MACSYMA's syntax, while remaining the same, would no longer denote fixed, pre-defined operations but would serve rather only as a convenient language for communicating the mathematical operations the user wants performed. With this view, a command or syntax could mean different things in different situations. For example, COEFF might mean RATCOEF in solving a quadratic but have its current definition in finding polynomial solutions to a polynomial equation; or F in DIFF(F,X) might mean the variable F if F has a value or the function F if it has a function definition. The input would be interpreted on the basis of not only the command line but also the user's plan. Similarly, the application of a rule would depend on not only the rule's pattern but also some notion of its use in achieving the user's goal. Where the system is unable to decide which of several interpretations the user prefers, it could inform him of the options rather than choosing a default as it does now. The essential idea again is to observe and use the structure of a user's session with MACSYMA to help ease his communication requirements. The implementation of such an apprentice could rely at the start on the programming apprentice technology described in (ref. 14).

Even if an apprentice were available, the user would still have to direct most manipulations of expressions. One frequently occurring type of manipulation is the application of several rules to some subpart of an expression. The SUBSTPART command was implemented for this purpose. However, the use of SUBSTPART requires a careful count of parts to select the desired subpart; if afterward the user wishes to apply another transformation, he must supply the part specification again; and of course all the intermediate expressions are saved. A better alternative is the use of a two-dimensional editor, a mechanism whereby the user is given control of a moveable "window" around an expression which he can zoom in on the desired subexpression using simple "up" and "down" commands, apply as many rules as he likes, then zoom out again to find the overall expression suitably modified. Such an editor would be much less tedious than the current SUBSTPART mechanism and would avoid the accumulation of unwanted intermediate results. A primitive 2D editor was programmed for MACSYMA by Richard Bryan but never released due to the inefficiency of the 2D display routines; with the current implementation, however, an efficient editor could be implemented.

In the long run the best solution to the subpart specification problem and the expression input problem is the graphics tablet. Technology has developed to the point where the recognition of hand-written expressions is feasible (refs. 15, 16). The remaining problem is inefficiency; however, with the advent of non-timeshared computers such as the LISP machine (ref. 17), the necessary processing need not be prohibitive. A less extreme alternative is the use of a light pen for 2D editing with keyboard input. A user could type in his expressions on the keyboard but move his window and cancel terms using a light pen. The disadvantage of either of these proposals is the limited availability of tablets and devoted processors at present. The LISP machine could, however, make the idea of "MACSYMA in a briefcase" a reality in a decade or so.

## 4. MACSYMA's Evolution

People sometimes complain that MACSYMA is difficult to understand or to control, and they usually cite specific properties of the system as primarily responsible, e.g. too many commands, too hard to specify subexpressions. These properties are not inherently difficulties but rather give rise to difficulties when the system is applied to certain tasks or by making the system difficult to understand or to use.

Such properties are not the results of poor design decisions. Rather, they are the best efforts of an active group of programmers to satisfy the conflicting goals of program modularity and efficiency and satisfaction of the user's needs (ref. 7). The resolution of this conflict is considerably harder for algebraic manipulation systems like MACSYMA than for more traditional programming languages. Most other programming language designs in a sense "define" the world in which they operate. MACSYMA's goal is to match as closely as possible a world that is already defined, namely mathematical manipulation as used in textbooks and on thousands of blackboards and notepads. Although some people say the constraints can and should be changed, with the current goal, they cannot be, even for a particularly elegant or well-structured design.

MACSYMA must also satisfy the often conflicting needs of a diverse user community. Many capabilities in MACSYMA were originally implemented to satisfy a particular need. As new users required analogous capabilities for other classes of expressions and in different environments, the capabilities had to be suitably broadened or refined. Viewed historically, MACSYMA is an excellent example of evolutionary programming. It is reminiscent of the progress of "normal science" described by Thomas Kuhn (ref. 18) in which a theory, or "paradigm", is repeatedly patched to repair its weaknesses until it is supplanted by a cognitively cleaner descendant. The growth of MACSYMA has led some people to believe that the new paradigm can be achieved only by avoiding the creation of new commands or by implementing simpler, more understandable evaluation algorithms. However, complexity in MACSYMA has usually resulted from the attempt to satisfy the conflicting needs of different users; if a new symbolic manipulation paradigm does arise, it will have to take these differing needs into account. The MACSYMA of the future will have to maintain an explicit, internal "model" of the user's goals and of his "plan" for achieving them.

## 5. Commentary

One of the purposes of this paper is to suggest some research projects oriented toward minimizing the difficulties of using a complex system like MACSYMA. Some of these projects are already underway. The MACSYMA Advisor is scheduled for limited release this summer. The new rational function representation is already partly implemented. The other projects are mentioned here to indicate some directions in which MACSYMA might go and to solicit implementation ideas and comment on their value.

The model for the "typical" MACSYMA user presented in section 2, on which the analysis of resource knowledge problems is based suffers two major deficiencies. The first is that it says little about domain dependent expertise. A sophisticated MACSYMA user probably mentally employs specialized procedural strategies and representations. The former are approximated by the templates in M, the latter are not dealt with at all. The model was designed to explain the performance of novice users as observed in several dozen protocols of MACSYMA usage; protocols of more advanced users were not included. The second major deficiency is that the model does not take learning into account. There is no sensitivity to how the user comes by his misconceptions. Also there is no information that could be used to determine how a consultant could best teach a point. It might, for example, be expedient to lie about something to make an explanation as simple as possible. These are several theories of learning in the literature (refs. 19, 20) that could be used in this regard.

The contributions of this paper are (1) its statement of the distinction between the various, essentially "orthogonal" types of difficulties of using a tool to help solve a problem and (2) its explanation of the function of user aids in meeting these difficulties, resulting in its proposal for more advanced aids based on this explanation. A learning difficulty arises when a system is too large or its primitives too complex for a new user to understand fully. A resource knowledge difficulty can arise whenever one is faced with a problem solving situation in a domain which one does not fully understand. The lack of knowledge may be incidental, as it is when the domain or device is fairly simple but time constraints make it impossible for the user to learn all that is necessary (e.g. using a calculator or oscilloscope). Or it may be essential, as when the domain is very complex and the user can't possibly learn everything (e.g. MACSYMA or business or law). Furthermore, the need is acute for computer systems like MACSYMA in which the level of commands is so close to the level of the task environment that the user is apt to confuse a simply defined procedure (like COEFF) with its mathematical counterpart (here coefficient) that it at best approximates. A communication difficulty can arise whenever a system's designer cannot provide every intended user with expertise tailored exclusively to his need. MACSYMA's knowledge based approach to algebraic manipulation drastically reduces communication difficulties; and by transforming MACSYMA from a programming language into a mathematician's apprentice, these difficulties might be even further reduced. Although the knowledge based approach engenders increased learning and resource knowledge difficulties these difficulties need not be prohibitive, if adequate user aids -- tutors and advisors -- are provided.

## References

1. Moses,J.: A MACSYMA Primer. Lab. Computer Sci., Massachusetts Inst. Technol., Oct. 1975.

2. Lewis,V.E.: User Aids for MACSYMA. Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper no. 28 of this compilation.)

3. Brown,J.S.;Burton,R.;and Bell,A.: SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting (An Example of AI in CAI). Rep. 2790, Bolt Beranek and Newman, Inc., 1974.

4. Burton,R.;and Brown,J.S.: A Tutoring and Student Modelling Paradigm for Gaming Environments. SIGCSE Vol. 8, No. 1, Feb. 1976, pp. 236-246.

5. Goldstein,I.P.;and Miller,M.: AI Learning Environments. Memo 389, AI Lab, Massachusetts Inst. Technol., Dec. 1976.

6. Genesereth,M.R.: Automated Consultation for Complex Computer Systems. Ph.D. Thesis, Harvard Univ., 1977.

7. Moses,J.: Algebraic Simplification: A Guide for the Perplexed. Commun. ACM Vol. 14, No. 8, Aug. 1971, pp. 527-537.

8. Mark,W.S.: The Reformulation Model of Expertise. TR-172, Lab. Computer Sci., Massachusetts Inst. Technol., Dec. 1976.

9. Davis,R.;Buchanan,B.;and Shortliffe,E.: Production Rules as a Representation for a Knowledge-Based Consultation Program. AIM-266, AI Lab, Stanford Univ., Oct. 1975.

10. Swartout,W.R.: A Digitalis Therapy Advisor With Explanations. TR-176, Lab. Computer Sci., Massachusetts Inst. Technol., Feb. 1977.

11. Mathlab Group: MACSYMA Reference Manual. Lab. Computer Sci., Massachusetts Inst. Technol., Nov. 1975.

12. Stoutemyer,D.: Qualitative Analysis of Mathematical Expressions Using Computer Symbolic Mathematics. Proceedings of the 1976 Symposium on Symbolic and Algebraic Computation, Aug. 1976, pp. 97-104.

13. Genesereth,M.R.: DB: A High Level Data Base System with Inference. Memo 4, MACSYMA Group, Massachusetts Inst. Technol., Dec. 1976.

14. Rich,C.;and Shrobe,H.E.: Understanding LISP Programs. Working Paper 82, AI Lab, Massachusetts Inst. Technol., Dec. 1974.

15. Martin,W.A.: Symbolic Mathematical Laboratory. TR-36, Lab. Computer Sci., Massachusetts Inst. Technol., Jan. 1967.

16. Williams,T.G.: On-Line Pasing of Hand-Printed Mathematical Expressions. NASA CR-1455, 1969.

17. Greenblatt,R.: The LISP Machine. Working Paper 79, AI Lab, Massachusetts Inst. Technol., Nov. 1974.

18. Kuhn,T.: *The Structure of Scientific Revolutions.* Univ. of Chicago Press, 1970.

19. Sussman,G.J.: *A Computational Model of Skill Acquisition.* Elsevier Pub. Co., 1975.

20. Winston,P.W.: Learning Stuctural Descriptions from Examples. TR-231, AI Lab, Massachusetts Inst. Technol., Sept. 1970.

# An Automated Consultant for MACSYMA*

Michael R. Genesereth

Center for Research in Computing Technology
Harvard University

Laboratory for Computer Science
Massachusetts Institute of Technology

Consider a person trying to solve a problem with a computer system he does not fully understand. And assume that, although he has encountered a difficulty due to his lack of knowledge, he is unwilling to learn more about the system than is necessary to solve the problem. The simplest way for him to acquire just the information he needs and no more is to consult an expert. Then, armed with the expert's advice, he may surmount the difficulty and solve the problem. A consultant is an information source that seeks to improve the user's model of its domain in "user-initiative" mode. Consultation is a method widely used in computer centers as well as in domains like business, law, and medicine. Unfortunately, human consultants are a scarce resource and quite expensive.

The purpose of this paper is to propose as an alternative an automated consultant, as exemplified by an "advisor" for the algebraic manipulation system MACSYMA. Such a program should be able to converse with its user in English about a difficulty he has encountered and provide information tailored to his need. The MACSYMA Advisor is a program distinct from MACSYMA with its own separate data base and expertise. However, for convenience the program can be called directly from MACSYMA and can access the user's data structures contained therein. The Advisor described here deals only with the "straight-line" or nested use of MACSYMA commands and not loops or user-defined functions.

The implementation of the Advisor relies heavily on an explicit, internal "model" of the user's state of knowledge, his goals, and his "plan" for achieving them. As a result, it can provide

---

* This work was supported, in part, by the United States Energy Research and Development Administration under Contract Number E(II-I)-3070 and by the National Aeronautics and Space Administration under Grant NSG 1323.

(This article is an extended abstract of a paper to be published in the proceedings of the Fifth International Joint Conference on Artificial Intelligence.)

more precise answers to a larger class of information needs than traditional user aids, such as manuals, information networks, and simple question-answering programs.

As a concrete example of the Advisor's performance, consider a scientist trying to solve a matrix eigenvalue problem using MACSYMA, as illustrated in figure I. An advisor episode is the connected fragment of discourse between a user and the Advisor that begins when the user types HELP() in MACSYMA and ends when he bids the Advisor goodbye. Each episode can have any number of subepisodes. A subepisode begins when the user asks a question and ends when the Advisor considers itself done. During a subepisode the Advisor may ask the user questions and expect answers; however, further questions by the user are ignored. In the protocol the first episode contains two subepisodes; the others one each, as marked.

```
(C1) (M[1,1]:2*Z,M[1,2]:Z,M[2,1]:2*Z-3,M[2,2]:Z-3)$
          ; The user tries to input his matrix by directly assigning to elements.
          ; Unbeknownst to him, this results in the creation of an array named M.
          ; In C2, he checks M and finds that his attempt failed.
(C2) M;
(D2)                               M

(C3) HELP()$
-------------------------------------------------------------------------
   | Advisor: Speak up!
   |
IA | User: How do I construct a matrix?
   |
   | Advisor: Use MATRIX or ENTERMATRIX.


   | User: What are the arguments to MATRIX?
   |
IB | Advisor: The rows of the matrix being constructed.
   |
   | User: Bye.
-------------------------------------------------------------------------
(C3) M:MATRIX([2*Z,Z],[3-2*Z,3-Z]);
                                       [   2 Z       Z   ]
(D3)                                   [                 ]
                                       [ 3 - 2 Z   3 - Z ]
          ; He enters his matrix. Note that the signs of two of the elements
          ; are different from before.
(C4) (M[1,1]-X)*(M[2,2]-X)-M[1,2]*M[2,1];
(D4)                    (Z - X - 3) (2 Z - X) - Z (2 Z - 3)
          ; Here he tries to compute the characteristic polynomial. The
          ; subscript notation is used for both array and matrix access. When
          ; an array exists, it means array selection, else matrix. Here is
          ; selecting from the array M not the matrix as he expects.
```

```
(C5) SOLVE(D4);
Error - more unknowns than equations.
          ; He tries to solve the polynomial, but forgets to specify the variable
          ; to solve for.  Not understanding the error message, he decides to
          ; solve it himself.
(C6) RATSIMP(D4);
                                               2
(D6)                         (- 3 X - 3) Z + X  + 3 X

(C7) (A:1,B:COEFF(D6,X,1),C:COEFF(D6,X,0))$
          ; Intending to use the quadratic formula, he picks out the coefficients
          ; of D6.  Unfortunately, by using a "$", he doesn't see that the answer
          ; he gets is not what he expects.
(C8) (-B+SQRT(B^2-4*A*C))/(2*A);
(D8)                                    0

(C9) C;
(D9)                                    0

(C10) HELP()$
--------------------------------------------------------------------------
```

User: In line C7, I used COEFF to get the zeroth coefficient of D6. The real coefficient is -3*Z, but MACSYMA returned 0!

Advisor: COEFF(EXP, VAR, POW) returns the correct coefficient of

2 $VAR^{POW}$ in EXP only if EXP is expanded with respect to VAR. Either expand first and then use COEFF or use RATCOEF.

User: Ok, thanks. Bye.

```
--------------------------------------------------------------------------
(C10) B:RATCOEF(D6,X,1);
(D10)                                3 - 3 Z

(C11) C:RATCOEF(D6,X,0);
(D11)                                 - 3 Z

(C12) (-B-SQRT(B^2-4*A*C))/(2*A);
                                                 2
                       - SQRT(12 Z + (3 - 3 Z) ) + 3 Z - 3
(D12)                  -----------------------------------
                                        2

(C13) EV(D12,Z=1);
(D13)                               - SQRT(3)
          ; Here he plugs in a value for Z to check the eigenvalue.
```

```
(C14) HELP()$
------------------------------------------------------------------------
```

User: D13 is wrong.

Advisor: What were you trying to do?

User: Find the eigenvalues of M.

Advisor: You used the determinant formula to compute the characteristic polynomial. You used RATSIMP to simplify it. You used the quadratic formula to solve it to obtain the eigenvalue. Am I right?

User: Yes.

Advisor: Did you intend the M[I,J] to select elements from the matrix M?

User: Obviously.

Advisor: When an array exists, the subscript notation means to select the corresponding element from the array. It works as matrix element selection only when no array exists. Were you aware you had an array?

User: No.

Advisor: An array is automatically created when an assignment is made to an unbound subscripted variable. Incidentally, to compute the characteristic polynomial of a matrix, you should use CHARPOLY.

User: Bye.

```
------------------------------------------------------------------------
(C14) CHARPOLY(M,X);
                                                     2
(D14)                          3 Z + X (- Z - 3) + X

(C15) FACTOR(D14);
(D15)                              - (X - 3) (Z - X)
          ; The user recomputes the characteristic polynomial and tries FACTOR
          ; from the result of which the roots are obvious.
(C16) LOGOUT()$
```

Fig. 1 - Protocol of MACSYMA use and consultation

The examples in this protocol were chosen to illustrate the most important of the Advisor's abilities. A full list of the types of questions it can answer follows.

(1) "What is the ... of ...?" Retrieval of a property of an object or concept given its name, e.g. subepisode 1B.

(2) "How do I do ...?" Retrieval of a command or method given a description of the task to be performed, subepisode 1A.

(3) "Is it the case that ...?" Evaluation of predicates.

(4) "Why is it the case that ...?" Ability to pinpoint a deficiency in the user's understanding and provide a precise answer, e.g. episodes 2 and 3.

(5) "How does MACSYMA do ...?" Procedural explanation of a result or fact.

Of these, the questions requiring the most sophisticated treatment are WHY and HOW. WHAT, HOWDO, and IS questions can be answered directly, with no consideration of the user's purpose or his state of knowledge. A WHY or HOW question calls for different answers to different people in different situations. The primary implementational contribution of this research is its technique for handling such questions and the data structures it uses.

Although the various parts of the Advisor have all been implemented, as of this writing they have not yet been combined into a working system. Also, the present data base is at best meager. The current timetable calls for its release to the MACSYMA user community this summer, where if successful it will find heavy use and provide valuable data for further improvements.

The important contributions of this research are (1) its recognition of the need for a consultant in any sufficiently complex domain and an indication of the nature of the user's needs, (2) a demonstration by design and partial implementation of the feasibility of automating such a consultant, (3) the model debugging algorithm utilizing a partial, explicit runtime model of the user and a partial plan for his behavior and based on an explicit design model. In general, a consultant is necessary whenever one is faced with (1) a problem solving situation (2) in a domain one does not fully understand. The lack of knowledge may be incidental, as it is when the domain or device is fairly simple but time constraints make it impossible for the user to learn all that is necessary (e.g. using a calculator or oscilloscope). Or it may be essential, as when the domain is very complex and the user can't possibly learn everything (e.g. MACSYMA or business or law). Furthermore, the need is acute for computer systems like MACSYMA in which the level of commands is so close to the level of the task environment that the user is apt to confuse a simply defined procedure (like COEFF) with its mathematical counterpart (here coefficient) that it at best approximates. It would be of interest to see whether an automated business or legal consultant could be constructed and how effective the techniques described here would be in those domains.

BLANK PAGE

BLANK PAGE

# A MACSYMA COMPUTER-ALGEBRA MOVIE DEMONSTRATION

David R. Stoutemyer
University of Hawaii

## ABSTRACT

The compelling excitment of using a powerful interactive computer-algebra system is hard to convey without a live demonstration, which is often impractical because of the size or location of an audience. However, a movie of a live demonstration is probably the next best way to convey the impact of interactive computer-algebra to an audience of newcomers. Sound projection 16mm equipment is far more available than the alternative of video tape equipment, which suffers from marginal resolution. Available from national educational film libraries and from the developers of computer-algebra systems, such films could significantly increase the awareness and utilization of this underutilized resource. To this end, I have produced a 10-minute prototype 8mm sound movie MACSYMA demonstration to show at this conference. While not of sufficient quality to be reproduced as a distributed 16mm film, it is hoped that this prototype will inspire a full-scale effort by someone with more cinematographic talent, with more funds, with access to high quality photographic resources, and with access to a fast terminal with high resolution.

BLANK PAGE

BLANK PAGE

316

# SOME MACSYMA PROGRAMS FOR SOLVING DIFFERENCE EQUATIONS*

John Ivie
University of California, Berkeley.

## INTRODUCTION

We describe here a set of programs to find closed-form solutions to linear recurrence relations (or "difference equations"), namely equations of the form

$$a_k\, u(n+k) + a_{k-1}\, u(n+k-1) + \ldots + a_0\, u(n) = g(n) \qquad (1)$$

where the coefficients $a_i$ are either constants (the constant coefficient case) or polynomials in $n$ (the variable coefficient case).

I would like to thank Richard Fateman for suggesting this problem to me, as well as for all of his help with the MACSYMA system.

## CONSTANT COEFFICIENT CASE

### The Characteristic Equation Method

We first consider the homogeneous case, that is when $g(n) = 0$ in equation (1) above. By substituting $x^{k-i}$ for $u(n+k-i)$ in equation (1), we obtain a polynomial equation; the solution to the recurrence relation can then be written as a linear combination of the roots of this polynomial. All of this is fairly easily done by means of the MACSYMA "SOLVE" command.

---

* This work was made possible by access to the MACSYMA system at M.I.T. , supported in part by ERDA under Contract Number E(11-1)-3070 and by NASA under Grant NSG 1323.

This is an extended abstract of a paper to appear in the ACM Transactions on Mathematical Software .

In the inhomogeneous case, when the right hand side of equation (1) is non-zero, we first find the homogeneous solution as above, and then add to it a particular solution of equation (1). This particular solution is found by the method of undetermined coefficients, which gives a set of linear equations to be solved via the "SOLVE" command. In our case here, we assume that g(n) is either a polynomial in n, a constant raised to a polynomial power, or sine or cosine of a linear function of n.

This method is implemented by the "CHAR" portion of our programs, which are given in an appendix.

## The Method of Generating Functions

This is another method for solving constant coefficient recurrence relations. This method finds the homogeneous and particular solutions at once, but is slower in our implementation than the characteristic equation method.

The basic idea of this method is the following: define the generating function $F(x)$ of the sequence $u(n)$ as

$$F(x) = \sum_{n=0}^{\infty} u(n)\, x^n \tag{2}$$

Using the recurrence relation (1), we can arrive at an algebraic equation for $F(x)$, so that $F(x)$ can be expressed as a rational function in x. We can then rewrite this rational function for $F(x)$ in terms of a partial fraction decomposition, so that the coefficients u(n) in F(x) can be identified, which gives the solution to the recurrence relation. (This technique is very much like a discrete Laplace transform). The main MACSYMA commands used to do all of this are "SOLVE" and "DIFF".

This method is implemented by the "GENF" portion of our programs.

## VARIABLE COEFFICIENT CASE

One method for solving variable coefficient recurrence relations is that of exponential generating functions. We assume that our generating function for the sequence $u(n)$ is of the form

$$Y(x) = \sum_{n=0}^{\infty} u(n)\; x^n / n! \tag{3}$$

Taking successive derivatives and using the recurrence relation (1), we obtain an ordinary differential equation for Y(x). Expanding the solution to the differential equation in a Taylor series, we see that the $n^{th}$ term of the series is the solution to our recurrence relation (1). This method can be programmed using the MACSYMA commands "ODE2" and "POWERSERIES" . This technique is implemented by the "VARC1" portion of our programs.

One major problem with this method is that there may be no way to find a closed-form solution to the differential equation which is obtained, or even to express a closed-form solution in a "nice" form. However, an explicit closed-form solution is available for first-order recurrence relations; this is implemented by "VARC2" in our programs. For second-order recurrences, a special check is made for those that can be solved in terms of Bessel functions; this is given by "BESSELCHECK" in our program listings.

## TESTING THE PROGRAMS

Using our programs, we were able to solve problems and examples taken from several textbooks ( as given in our list of references ). The following is a small sample of some typical problems:

```
(C66) CHAR(U(N+1)-U(N),(1/6)*N*(N-1)*(N-2)+N-1,U,N,1,[U(0)=1]);
                                   3    2
                                  N    N    23 N   7
(D71)                   U(N) = N (-- - -- + ---- - -) + 1
                                  24   4     24    4

(C72) CHAR(U(N+2)-2*U(N+1)+U(N),N**2,U,N,2,[U(0)=0,U(1)=1]);
                                      2
                                  2  N    N   5     5 N
(D77)                     U(N) = N  (-- - - + --) + ---
                                     12   3   12      6

(C78) GENF(U(N+2)-U(N),2**N,U,N,2,[U(0)=1,U(1)=0]);
                                      N          N
                                     2    2 (- 1)
(D84)                         U(N) = -- + --------
                                     3       3

(C85) CHAR(U(N+2)-4*U(N),3+2*N,U,N,2,[U(0)=1,U(1)=0]);
                                 N           N
                              7 2    25 (- 2)    2 N   13
(D90)                 U(N) = ---- + --------- - --- - --
                               4       36        3     9
```

```
(C43)  VARC1(U(N+1)-(N+1)*U(N),0,U,N,1,[U(0)=1]);
(D44)                              U(N) = N!

(C40)  VARC1(U(N+1)-(N+1)*U(N),1,U,N,1,[U(0)=1]);
                                           N
                                          ====
                                          \       1
(D41)                        U(N) = N!     >      ---
                                          /      I6!
                                          ====
                                          I6 = 0

(C42)  VARC2(U(N+1)-(N+1)*U(N),1,U,N,1,[U(0)=1]);
                           N - 1            N
                           /===\           ====
                            ! !            \           1
(D42)              U(N) = (  ! !  (I + 1))  >     -------------
                            ! !            /     I - 1
                           I = 1           ====  /===\
                                           I = 0  ! !
                                                  ! !  (J + 1)
                                                  ! !
                                                 J = 1

(C12) VARC1(U(N+2)-(3*N+2)*U(N+1)+5*U(N),0,U,N,2,[U(0)=0,U(1)=1]);
(D12)              A LINEAR COMBINATION OF BESSEL FUNCTIONS
```

Using the 77 problems from the references which we tried, we found that CHAR had an average running time of 585 msec., while that for GENF was 1113 msec. . Thus, the characteristic equation method is much faster in our implementation here.

## CONCLUDING REMARKS

After this paper was written, we became aware of a similar paper by Cohen and Katcoff (to appear in Transactions on Mathematical Software). Their methods seem somewhat more general (they deal with systems also); however, our programs are much shorter and seem to have faster running times.

## REFERENCES

1. Anderson, Ian: A First Course in Combinatorial Mathematics. Oxford Univ. Press,Inc., 1974.

2. Brand, Louis: Differential and Difference Equations. John Wiley & Sons, Inc., 1966.

3. Goldberg, Samuel: Introduction to Difference Equations. John Wiley &Sons,Inc., 1958.

4. Hall, Marshall, Jr.: Combinatorial Theory. Blaisdell Pub. Co., 1967.

5. Liu, C.L.: Introduction to Combinatorial Mathematics. McGraw-Hill Book Co.,Inc., 1968.

## APPENDIX

For completeness, we give here a listing of the actual MACSYMA code for our programs.

```
     /*THIS BLOCK CHECKS FOR A POLYNOMIAL IN N*/

POLYP(G,N):=BLOCK([D,F,C],
     G:RATEXPAND(G), IF FREEOF(N,G) THEN RETURN(TRUE),
     D:HIPOW(G,N), F:TRUE,
     FOR I:D STEP -1 THRU 0 DO
     (C:COEFF(G,N,I), IF NOT(FREEOF(N,C)) THEN F:FALSE,
      G:RATEXPAND(G-C*N**I)),
     RETURN(IS(G=0 AND F)))$

     /*THIS BLOCK CHECKS FOR A CONSTANT TO A POLYNOMIAL POWER*/

POLYINN(X,N):=BLOCK([B,E],
     IF INPART(X,0)="*" THEN
     RETURN(POLYINN(INPART(G,1),N) AND POLYINN(INPART(G,2),N)),
     IF INPART(X,0)#"**" THEN RETURN(FALSE),
     B:INPART(X,1),
     E:INPART(X,2),
     IF NOT FREEOF(N,B) THEN RETURN(FALSE),
     RETURN(POLYP(E,N)))$
     /*THIS BLOCK IMPLEMENTS THE CHARACTERISTIC EQUATION METHOD*/

CHAR(E,G,U,N,K,IV):=BLOCK([GENSOL,HOMSOL,PARSOL,LOS,MULTIPLICITIES,
H,V,L,SS,DISPFLAG],
LOCAL(A,AA,B,R,M),
DISPFLAG:FALSE,
  FOR I:0 THRU K DO
    AA[I]:COEFF(E,U(N+K-I)),
    H:0,
  FOR I:0 THRU K DO
    H:H+AA[I]*U(N+K-I),
  IF H#E THEN RETURN("ERRONEOUS INPUT"),

  FOR I:0 THRU K DO
    H:SUBST(U**(K-I),U(N+K-I),H),

  MULTIPLICITIES:TRUE,
  LOS:SOLVE(H,U),
  FOR I:1 THRU LENGTH(LOS) DO
    (R[I]:LOS[I], R[I]:RHS(EV(R[I]))),
     M[I]:MULTIPLICITIES[I]),
  HOMSOL:
    SUM(SUM(A[I,J]*N**(M[I]-J),J,1,M[I])*R[I]**N,I,1,LENGTH(LOS)),


  IF G=0 THEN
      (V:[ ],
       FOR I:1 THRU LENGTH(LOS) DO
       FOR J:1 THRU M[I] DO V:CONS(A[I,J],V),
          L:[ ],
       FOR Q:0 THRU K-1 DO L:CONS(SUBST(Q,N,HOMSOL)=U(Q),L),
       SS:EV(SOLVE(L,V),IV),
       RETURN(U(N)=(EV(HOMSOL,SS))))

  ELSE IF POLYP(G,N) = TRUE THEN
      (G:RATEXPAND(G), PARSOL:SUM(B[J]*N**J,J,0,HIPOW(G,N)),
       FOR J:0 THRU K DO
        (L:0, V:E,
          FOR I:0 THRU K DO
          (L:RATEXPAND(SUBST(N+K-I,N,B[J]*N**J)),
           V:RATEXPAND(SUBST(L,U(N+K-I),V))),
           V:RATSIMP(V),
            IF V#0 THEN RETURN(V) ELSE PARSOL:N*PARSOL),
          V:E,
       FOR I:0 THRU K DO (L:RATEXPAND(SUBST(N+K-I,N,PARSOL)),
          V:RATEXPAND(SUBST(L,U(N+K-I),V))),
          L:[ ],
       FOR I:0 THRU HIPOW(PARSOL,N) DO
          L:CONS(COEFF(V=G,N,I),L),
          V:[ ],
    FOR J:0 THRU HIPOW(PARSOL,N) DO
       V:CONS(B[J],V),
    SS:SOLVE(L,V),
    PARSOL:EV(PARSOL,SS))

ELSE IF POLYINN(G,N) = TRUE THEN
   (PARSOL:B1*G,
    FOR J:0 THRU K DO
     (L:0, V:E,
       FOR I:0 THRU K DO
       (L:SUBST(N+K-I,N,PARSOL), V:SUBST(L,U(N+K-I),V)),
        V:RATSIMP(V),
        IF V#0 THEN RETURN(V) ELSE PARSOL:N*PARSOL),
    SS:SOLVE(V=G,B1),
    PARSOL:EV(PARSOL,SS))

ELSE IF INPART(G,0)=SIN OR INPART(G,0) = COS THEN
   (PARSOL:B[1]*SIN(INPART(G,1)) + B[2]*COS(INPART(G,1)),
    FOR J:0 THRU K DO
     (L:0, V:E,
       FOR I:0 THRU K DO
       (L:EXPAND(SUBST(N+K-I,N,PARSOL)),
        V:EXPAND(SUBST(L,U(N+K-I),V))),
    V:TRIGEXPAND(V),
    IF V#0 THEN RETURN(V) ELSE PARSOL:N*PARSOL),
       V:E,
    FOR I:0 THRU K DO(L:EXPAND(SUBST(N+K-I,N,PARSOL)),
       V:EXPAND(SUBST(L,U(N+K-I),V))),
    V:TRIGEXPAND(V),
       L:[ ],
    LT:[SIN(INPART(G,1)),COS(INPART(G,1))],
    FOR JJ:1 THRU 2 DO
       L:CONS(COEFF(V=G,LT[JJ]),L),
       V:[ ],
    FOR J:1 THRU 2 DO
       V:CONS(B[J],V),
    SS:SOLVE(L,V),
    PARSOL:EV(PARSOL,SS))


ELSE RETURN("CAN'T BE SOLVED IN CLOSED FORM BY PROGRAM"),

GENSOL:HOMSOL + PARSOL,
   V:[ ],
FOR I:1 THRU LENGTH(LOS) DO
FOR J:1 THRU M[I] DO V:CONS(A[I,J],V),
   L:[ ],
FOR Q:0 THRU K-1 DO
L:CONS(SUBST(Q,N,GENSOL)=U(Q),L),
SS:EV(SOLVE(L,V),IV),
RETURN(U(N)=(EV(GENSOL,SS))))$
```

```
     /*THIS BLOCK IMPLEMENTS THE GENERATING FUNCTION METHOD*/

GENF(E,G,U,N,K,IV):=BLOCK([MULTIPLICITIES,L,V,SS,VV,LOS,
NR,F,SOL,P,DISPFLAG],
LOCAL(A,AA,B),
DISPFLAG:FALSE,
  FOR I:0 THRU K DO
    AA[I]:COEFF(E,U(N+K-I)),
    H:0,
  FOR I:0 THRU K DO
    H:H+AA[I]*U(N+K-I),
  IF H#E THEN RETURN("ERRONEOUS INPUT"),

     L:E,
  FOR I:0 THRU K DO
    L:SUBST((F-SUM(U(J)*X**J,J,0,K-I-1))*X**I,U(N+K-I),L),

  IF G=0 THEN
     (S:SOLVE(L,F),
      F:EV(F,S))

  ELSE IF POLYP(G,N) = TRUE THEN
     (G:RATEXPAND(G),
      V:SUBST(X**K/(1-X)*COEFF(G,N,0),COEFF(G,N,0),G),
      VV:RATSIMP(DIFF(1/(1-X),X)),
      FOR I:1 THRU HIPOW(G,N) DO
         (V:SUBST(X**K*X*VV*COEFF(G,N,I),COEFF(G,N,I)*N**I,V),
          VV:RATSIMP(DIFF(X*VV,X))),
      V:RATSIMP(V),
      SS:SOLVE(L=V,F),
      F:EV(F,SS))

  ELSE IF POLYINN(G,N) = TRUE AND HIPOW(INPART(G,2),N) < 2 THEN
     (G1:(X**K)*(INPART(G,1)**COEFF(INPART(G,2),N,0)),
      G2:1 - X*(INPART(G,1)**COEFF(INPART(G,2),N,1)),
      V:RATSIMP(G1/G2),
      SS:SOLVE(L=V,F),
      F:EV(F,SS))

  ELSE RETURN("CAN'T BE SOLVED IN CLOSED FORM BY PROGRAM"),


  MULTIPLICITIES:TRUE,
  LOS:SOLVE(NEWRAT(F),X),
  FOR I:1 THRU LENGTH(LOS) DO
     (R[I]:LOS[I], R[I]:RHS(EV(R[I]))),
      M[I]:MULTIPLICITIES[I]),
      V:[ ],
   B:PRODUCT((1-R[I]*X)**M[I],I,1,LENGTH(LOS)),
   FOR I:1 THRU LENGTH(LOS) DO
   FOR J:1 THRU M[I] DO
      (P[I,J]:B*A[I,J]/((1-R[I]*X)**J),  V:CONS(A[I,J],V)),
   P:SUM(SUM(P[I,J],J,1,M[I]),I,1,LENGTH(LOS)),

      L:[ ],
   NF:RATEXPAND(NUM(F)/ABS(COEFF(DENOM(F),X,0))), P:RATEXPAND(P),
   FOR I:0 THRU HIPOW(RATEXPAND(B),X)-1  DO
      L:CONS(COEFF(NF=P,X,I),L),
   SSS:EV(SOLVE(L,V),IV),

   SOL:SUM(SUM(A[I,J]*COEFF(DENOM(F),X,0)/ABS(COEFF(DENOM(F),X,0))*
      BINOMIAL(J+N-1,N)*R[I]**N,J,1,M[I]),I,1,LENGTH(LOS)),

   RETURN(U(N)=(EV(SOL,SSS))))$

      /*THIS BLOCK FINDS THE NEW POLYNOMIAL ASSOCIATED TO F*/

NEWRAT(F):=BLOCK([HD,CP,DP],
      HD:HIPOW(DENOM(F),X),
      CP:COEFF(DENOM(F),X,HD),
      DP:SUM((COEFF(DENOM(F),X,I))/CP*X**I,I,0,HD),
      RETURN(SUM(COEFF(DP,X,HD-I)*X**I,I,0,HD)))$

      /*THIS BLOCK IMPLEMENTS THE VARIABLE COEFFICIENT METHOD*/

VARC1(E,G,U,N,K,IV):=BLOCK([V,VV,EQ,Y,CAUCHYSUM,FINSOL,SERSOL,DISPFLAG],
LOCAL(A,B),DISPFLAG:FALSE,
   FOR I:0 THRU K DO
     (A[I]:COEFF(E,U(N+I)),
      A[I]:RATEXPAND(A[I]),
      IF POLYP(A[I],N)=FALSE THEN RETURN("CAN'T DO IT")),
   IF K=2 AND (B:BESSELCHECK(E,K) # FALSE) THEN RETURN(B),
   V:RATEXPAND(E),
   FOR I:K STEP -1 THRU 0 DO
      FOR J:HIPOW(A[I],N) STEP -1 THRU 0 DO
        (V:RATSUBST(X**J*'DIFF(Y,X,I+J),N**J*U(N+I),V),
         V:RATEXPAND(V)),
   V:RATSUBST(Y,'DIFF(Y,X,0),V),
   V:RATEXPAND(V),
   IF POLYP(G,N) = TRUE THEN
     (G:RATEXPAND(G), VV:G,
      FOR I:0 THRU HIPOW(G,N) DO
         VV:SUBST(X**I,N**I,VV),
         VV:%E**X*VV)
   ELSE RETURN("CAN'T DO IT"),
      EQ:V-VV,
   DEPENDENCIES(Y(X)),
   SOL:ODE2(EQ=0,Y,X),
   IF K=1 THEN FINSOL:INITIAL1(SOL,X=0,Y=EV(U(0),IV))
   ELSE IF K=2 THEN FINSOL:IC(SOL,X=0,Y=EV(U(0),IV),'DIFF(Y,X)=EV(U(1),IV)
   ELSE RETURN("O.D.E. CAN'T BE SOLVED AT PRESENT BY MACSYMA"),
      CAUCHYSUM:TRUE,
   SERSOL:POWERSERIES(RHS(FINSOL),X,0), SERSOL:EXPAND(SERSOL),
   IF ATOM(SERSOL) THEN RETURN("U(N)=0 FOR N > 0"),
   B:INPART(SERSOL,1),
   B:EV(B,X=1),
   IF ATOM(B)=FALSE THEN B:SUBSTPART(N,B,4),
   RETURN(U(N)=((N!)*B)))$

      /*THIS BLOCK CHECKS FOR A BESSEL RECURRENCE RELATION*/

BESSELCHECK(E,K):=BLOCK([A,ANS],
LOCAL(A),
      FOR I:0 THRU K DO
      (A[I]:COEFF(E,U(N+I)),
       A[I]:RATEXPAND(A[I])),
      IF NOT(INTEGERP(A[0])) THEN RETURN(FALSE),
      IF NOT(INTEGERP(EV(A[1],N=0))) THEN RETURN(FALSE),
      IF NOT(HIPOW(A[1],N)=1) THEN RETURN(FALSE),
      IF NOT(INTEGERP(COEFF(A[1],N,1))) THEN RETURN(FALSE),
      IF NOT(A[2]=1) THEN RETURN(FALSE),
      ANS:"A LINEAR COMBINATION OF BESSEL FUNCTIONS",
      /*EXACT DETAILS ARE OF NO SIGNIFICANCE,SINCE WE ARE MERELY
        DEMONSTRATING THE FEASIBILITY OF THIS APPROACH*/
      RETURN(ANS))$

       /*THIS BLOCK IMPLEMENTS THE FIRST ORDER METHOD*/

VARC2(E,G,U,N,K,IV):=BLOCK([H,P,V,C,SOL],
LOCAL(AP,P),
      P:(-1)*COEFF(E,U(N))/COEFF(E,U(N+1)),
      V:G/COEFF(E,U(N+1)),
      S[J]:SUBST(J,N,P),
      S[I]:SUBST(I,N,P),
      P[N]:PRODUCT(S[I],I,1,N-1),
      H[I]:SUBST(I,N,V)/PRODUCT(S[J],J,1,I-1),
      V1:SUM(H[I],I,0,N),
      AP:EV(U(0)-SUBST(0,N,V),IV),
      RETURN(U(N)=AP*P[N]+P[N]*V1))$
```

33

# SOME COMMENTS ON SERIES SOLUTIONS *

Richard J. Fateman
University of California, Berkeley

## 1. SUMMARY

The use of power series and truncated power series in the MACSYMA system for algebraic manipulation is illustrated. Algebraic and differential equations are solved using Taylor series or asymptotic series. Deficiencies of the current scheme are noted, and remedies suggested.

## 2. Infinite Power Series

The general term "series" is used for at least two different types of expressions in MACSYMA (ref. 1). A power series, informally, is an exact representation of a function usually of one complex variable, f(z), sometimes requiring the summation of an infinite number of terms, where the power series may converge only for |z|<R, where R is the radius of convergence. Examples:

```
exp(x)= sum(x^i/i!,i,0,inf), convergent for |x| < inf;

x+3*x^3=sum(a[i]*x^i,i,0,inf) where a[1]=1,  a[3]=3,
       a[0]=a[2]=a[j] = 0  j≥4
(or more compactly, x+3*x^3)  convergent for |x| < inf;

1/(1-x) = sum(x^i,i,0,inf)  convergent for |x| < 1;
```

These are power series expansions about x=0. Translation to a point $a \neq 0$ is trivially accomplished for a finite series: $x+3*x^3$ ==> $a^3+a + (9*a^2+1)*(x-a) + 9*a*(x-a)^2 + 3*(x-a)^3$. For a function f(z) analytic at a finite point c, a linear transformation can be used to map the point c to the origin. Expansion about a pole of f(z) in the complex plane is sketched in section 5. Such problems are examined in a mathematical context in numerous texts of which references 2-3 are examples.

---

*. The work described herein was performed with the help of MACSYMA, which is supported, in part, by the United States Energy Research and Development Administration under Contract Number E(11-1)-3070 and by the National Aeronautics and Space Administration under Grant NSG 1323.

Power series as used in MACSYMA need not consist solely of non-negative exponents: $exp(x)/x = sum(x^{i}/(i+1)!,i,-1,inf)$.

They need not consist solely of integer exponents: $exp(x)*x^{1/3} = sum(x^{i+1/3}/i!,i,0,inf)$.

The existence of power series solutions to various types of equations, (typically differential equations) has been established, (see, for example, ref. 3) but proofs, even if constructive, rarely provide a means for expressing in closed form, in terms of some limited class of functions and forms, the power series itself. By "forms" we mean summations, products, or integrals with finite or infinite limits, or derivatives of finite order of known functions.

To be more precise, in terms of finite presentation, a univariate power series is a triple: $(x, \{I_k\}, \{a_k\})$. The first item, x, is the independent variable (indeterminate) of the series, $\{I_k\}$ is a sequence of exponents, and $\{a_k\}$ is a sequence of coefficients. Usually the sequences are infinite, and therefore cannot be represented in a computer by enumeration, but rather by generation. It is convenient to require that given some value from $\{I_k\}$, say j, the corresponding k such that $I_k = j$ can be found: this is the operation of finding out the coefficient of a given power of x.

MACSYMA produces power series via the POWERSERIES command in a closely related form. The triple specified above is only a slight generalization of the representation: the summation form used in MACSYMA devolves down to a subset of the integers, and thus the exponents are a function of the index rather than members of the exponent set.

Furthermore, the MACSYMA default result for the product of two infinite series $sum(a_i*x^{i},i,0,inf)$ and $sum(b_i*x^{i},i,0,inf)$ has the form $sum(sum(a_i*b_j*x^{i+j},j,0,inf),i,0,inf)$ rather than (with CAUCHYSUM:TRUE) $sum(sum(a_j*b_{k-j},j,0,k)*x^{k},k,0,inf)$ in which the coefficient of $x^k$ is a finite sum. If the conversion to "Cauchy"-style products were the only barrier, then there would be little cause for alarm. Much more difficult is the generation of an explicit form for composition. Although implicit forms, usually recurrence relations for the sequence $\{a_i\}$, can be calculated, these do not satisfy our "finite closed form" restriction.

Thus while infinite power series are a powerful mathematical construction, operations on them may lead outside the domain of series with explicit finitely generated terms.

This is not to say this leads necessarily to intractable problems: on the contrary, we can say the same thing about trigonometric or algebraic functions (square roots for example) since they may lead from the finitely-generated rational numbers to algebraic or transcendental numbers.

Nevertheless, if one is attempting to compute with power series, it is useful to minimally ensure that the ratio test for convergence can be computed for any power series expression: $lim(a_n/a_{n-1},n,inf) < inf$, where $a_n$ is the coefficient of $x^n$. The finitely-generated restriction gives one a good possibility for this, although it is not a necessary condition for the power-series ratio test to be computable.

## 3. Truncated Power Series

The second type of series construction in MACSYMA which by and large ignores questions of ultimate convergence, but has considerable advantage in terms of ease of computation, is the truncated power series (TPS- so called in ALTRAN and SCRATCHPAD) or the "Taylor Series" form in MACSYMA. Since it is unreasonable to restrict our discussion to Taylor series (no negative exponents), and the name used in MACSYMA is primarily of historical origin, we will use the phrase truncated power series or TPS to denote this type of expression. A TPS is a finite subset of the coefficient-exponent pairs in a full power series. The representation includes an indication of the order of truncation which has been imposed by the user and/or the system. In some cases the order of truncation is altered by operations, which include all rational operations (where division by TPS with a zero constant term may lead to a truncatedLaurent series with negative exponents). Additional operations such as power series reversion, multivariate expansions, a type of asymptotic expansion, and extension to more terms are described in the MACSYMA manual (ref. 1, also see ref. 4 for a more detailed discussion of univariate TPS in MACSYMA). Other systems offering automated handling of TPS include ALTRAN and SCRATCHPAD (refs. 5,6). Facilities are present in many earlier algebra systems for handling "weighted variables" but it seems that only recently has an appreciation developed for the fact that these rudimentary power-series ideas are easily generalized to operations such as inversion and reversion.

We indicate in passing that asymptotically fast methods of computation on TPS have been described by Brent and Kung (ref. 7), Kung and Traub (ref. 8), to replace the classical methods (see, for example, Lipson (ref. 9) or Knuth (ref. 10)). For the remainder of this paper we will be concerned with the use of TPS in the solution of equations, and the relative rapidity of the algorithms underlying the methods will not much affect the usefulness of the results.

For our purposes, we choose to omit from the TPS repertoire a number of the more sophisticated features. We consider a TPS to be identical with a power series with the change that $\{I_k\}$, the set of exponents, is necessarily finite (and a prefix of the infinite set), and each operation on TPS must preserve as many terms in the answer as can be guaranteed correct, given the operand description. In some cases additional assumptions are made. For example, given the TPS $Y = 1-x+\ldots$, then $1/Y = 1+x+\ldots$ Yet if $Y$ is in fact $\ldots + 1 - x + \ldots$, e.g. $1/x + 1 - x + \ldots$ then $1/Y = x - x^2 + \ldots$, rendering even the constant term incorrect. Thus we will assume, except when explicitly stated otherwise, that all negative exponent terms are given.

## 4. Algebraic Equations and Truncated Power Series

If we lay aside cautions concerning the validity of expressing an unknown function as a TPS, we can often proceed to find the coefficients in the series by substitution into a defining equation. We illustrate with examples of algebraic equations and differential equations. Additional examples can be demonstrated combining these two, or adding the operation integration.

The techniques in this section are not intended to be general prescriptions for all problems of this nature, but to illustrate a common-sense approach which frequently is useful.

Consider the following irreducible cubic equation:

$L^3 - e*(L+1) = 0.$

The three roots for L obtainable by means of the cubic formula are, as expected, unwieldy. If we assume the existence of a solution $L(e) = \mathrm{sum}(LL_i*e^i,i,0,inf)$ and try to determine $\{LL_i\}$ by substitution, we find that setting to zero coefficients of various powers of e in the equation result in inconsistencies (e.g. $-1 = 0$). A few moments consideration of the defining equation suggests that such a series does not exist, but that if we solve for $L^3$, then a cube root of the lowest term in e, ($e^1$) will provide a basis for expansion. In fact, substitution of $e^3$ for e in the original equation (or alternatively, expansion of L in terms of the cube-root of e), serves the purpose precisely.

Now that the general form has been chosen there are several levels of generality in which the coefficients may be found.

The infinite power series approach, namely to substitute power series forms into the defining equation and solve for the arbitrary coefficients in closed form as a function of n, the index of $e^{1/3}$, would be the most powerful. Unfortunately, MACSYMA cannot do this automatically, although with sufficient prompting part of the algebra can be accomplished. (It would be interesting to completely characterize what can be done by mechanical means to find closed form solutions; the result would be analogous to the Risch integration algorithm.)

Less satisfactory, but more to be expected considering the small set of solvable recurrences, is the derivation of a recurrence which can be marched to any desired order.

Yet more likely is that a set of equations can be generated such that all $LL_i$ up to some fixed $i = N$ can be found. Of course it may happen that the defining equations for the coefficients are no more tractable than the original equation. This is certainly possible for algebraic equations but if we start with a differential equation we have at least traded it for an algebraic problem.

Elementary arithmetical considerations suggest that polynomial equations of the form $L^n - e*$(lower order terms in L) $= 0$ have formal power series solutions for L in terms of $e^{1/n}$. In fact the degree of the smallest non-zero term can be predicted. A complete procedure for such determinations for algebraic expressions would be interesting, but in general we must tackle rather difficult problems: The computation of $LL_0$ in $L = \mathrm{sum}(LL_i*e^i,i,0,inf)$ given a defining polynomial in L is in general as hard as (and may be the same as) finding an algebraic expression for L itself. If e is missing from the equation, then trivially $L = LL_0$.

Some algebraic equations can be dealt with in a very powerful framework involving "Newton-like" iterations. (ref. 8) Rather than use these somewhat esoteric methods here, we will proceed on a more direct path to specific examples. Section 7 treats Newton iterations briefly. The results coincide when both approaches are appropriate.

As an illustration of the algebraic substitution technique on the example given above, we present the following dialogue with MACSYMA. The definition of the function SOLVEALL is more complex than need be, perhaps, for this simple function, but it illustrates the "blind" use of this substitution technique. In this particular case, $LL_0$ is found from the coefficient of $e^0$, $LL_1$ is found from the coefficient of $e^3$ (and is chosen arbitrarily to be one of the three roots of $LL_1^3-1=0$), and $LL_2$ through $LL_4$ are determined from the coefficients of $e^4$ through $e^6$. While $LL_5$ and $LL_6$ appear in the equation, their values are not determined because the appropriate coefficients are already zero.

```
(C1) EQ:  L^3-E*(L+1);
                            3
(D1)                       L  - E (L + 1)

(C2) DEFTAYLOR(H(E),SUM(LL[I]*E^I,I,0,INF));
(D2)                                [H]

(C3) TAYLOR(SUBST(H(E),L,EQ),E,0,4);
          3             2
(D3)/T/ LL   + (3 LL  LL   - LL   - 1) E
          0         1   0      0

            2            2          2
 + (3 LL  LL   + 3 LL  LL   - LL ) E
        2   0        1   0      1

            2                       3          3
 + (3 LL  LL   + 6 LL  LL  LL   + LL   - LL ) E
        3   0        2   1   0      1      2

            2                      2                 2          4
 + (3 LL  LL   + (6 LL  LL   + 3 LL ) LL   + 3 LL  LL   - LL ) E  + . . .
        4   0         3   1        2    0        2   1      3
```

Note that the first three coefficients imply that $LL_0=0$, $LL_1=-1/3$, and $LL_1=0$ simultaneously, clearly inconsistent.

```
(C4) EQ3:SUBST(E^3,E,EQ);
                            3    3
(D4)                       L  - E  (L + 1)

(C5) RES:TAYLOR(SUBST(H(E),L,EQ3),E,0,6);
          3            2              2        2       2
(D5)/T/ LL   + 3 LL  LL  E + (3 LL  LL   + 3 LL  LL ) E
          0        1   0          2   0        1   0
```

```
           2                     3             3
+ (3 LL  LL  + (6 LL  LL  - 1) LL  + LL  - 1) E
       3   0        2   1        0     1

           2                    2                2         4
+ (3 LL  LL  + (6 LL  LL  + 3 LL ) LL  + 3 LL  LL  - LL ) E
       4   0        3   1       2    0       2   1     1

           2                                         2           2
+ (3 LL  LL  + (6 LL  LL  + 6 LL  LL ) LL  + 3 LL  LL  + 3 LL  LL
       5   0        4   1       3   2    0       3   1       2   1

        5            2                                2
- LL ) E  + (3 LL  LL  + (6 LL  LL  + 6 LL  LL  + 3 LL ) LL
    2            6   0        5   1       4   2       3    0

          2                     3         6
+ 3 LL  LL  + 6 LL  LL  LL  + LL  - LL ) E  + . . .
      4   1       3   2   1     2     3
```

The value of RES above is the result of substituting a series into EQ3, the same as EQ, but with E replaced by $E^3$. We now define a fairly general program to solve for all the coefficients in such a defining equation. The program below is not asymptotically fast, since examples can be concocted for which it is O(n!) for n terms desired. Iterative methods described in section 6 provide the potential for much faster construction of terms, yet the relative simplicity of **SOLVEALL** below -- in not relying on how the equation was generated, is attractive.

```
(C6) /* SOLVE EQ FOR CC[0] . CC[LIM] AS REQUIRED TO MAKE EQ(X) ZERO. */

SOLVEALL(EQ,X,CC,LIM):=
BLOCK([C,VARS,S,K],
K:0,
WHILE EQ#0 AND K<LIM  DO
 (C:COEFF(EQ,X,K),
  IF C#0 THEN
   (VARS:LISTOFVARS(C),
    UNK:MINF,/* MINUS INFINITY */
    FOR I IN VARS DO
        IF NOT(ATOM(I)) AND PART(I,0)=CC AND PART(I,1)>UNK THEN
         UNK:PART(I,1),
          /*PICK OUT HIGHEST INDEX */
    IF UNK = MINF  THEN ERROR("INCONSISTENT"),
      /* NO WAY TO MAKE COEFF. ZERO */
    UNK:CC[UNK],
    S:SOLVE(C,UNK),
    IF S =[ ] THEN ERROR("INCONSISTENT")
        ELSE (IF REST(S)#[ ] THEN PRINT
                    ("MULTIPLE SOLUTIONS: FIRST ONE CHOSEN"),
            UNK::RHS(EV(S[1])), /* ASSIGN COEFFICENT VALUE */
            EQ:EV(EQ))),
  K:K+1))$
```

```
(C7) SOLVEALL(RES,E,LL,6);
SOLUTION

(E7)                             LL  = 0
                                   0
MULTIPLICITY 3
SOLUTION

                                 %I SQRT(3) - 1
(E8)                      LL  = --------------
                            1         2

                                 %I SQRT(3) + 1
(E9)                      LL  = - --------------
                            1           2

(E10)                            LL  = 1
                                   1
MULTIPLE SOLUTIONS: FIRST ONE CHOSEN
SOLUTION

                                 %I SQRT(3) + 1
(E11)                     LL  = - --------------
                            2           6
SOLUTION

(E12)                            LL  = 0
                                   3
(D12)                             DONE
```

The difficulty with multiple solutions for $LL_1$ can be nicely resolved in MACSYMA as follows: Let w be a primitive root of $w^3$-1 (i.e. a root of the irreducible factor of $w^3$-1 with roots which generate all distinct cube roots of 1: $w^2$+w+1), remove old values of LL, and then set $LL_1$ to w. SOLVEALL then uses the given value for $LL_1$, and proceeds to find the other coefficients. By informing MACSYMA via TELLRAT and ALGEBRAIC about the special properties of w, $LL_2$ come out nicely reduced.

```
(C13) (KILL(LL),TELLRAT(W^2+W+1),
/* W IS PRIMITIVE CUBE-ROOT OF 1 */
  LL[1]:W, ALGEBRAIC:TRUE)$

(C14) SOLVEALL(RES,E,LL,6);
SOLUTION

(E14)                           LL  = 0
                                  0

MULTIPLICITY 3
SOLUTION

                                     W + 1
(E15)                        LL  = - -----
                               2       3

SOLUTION

(E16)                           LL  = 0
                                  3
```

## 5. Differential Equations and Truncated Power Series

This section deals with an admittedly trivial differential equation as an illustration. We demonstrate the types of operations supplied by MACSYMA and how to use them. The differential equation (assume right hand side is zero) is entered on line C17, we remove the previous values for the LL-array, and generate the TAYLORSOL as given below.

```
(C17) DE:DIFF(H(E),E,2)-A^2*H(E);
                           2
                          d           2
(D17)                     --- H(E) - A  H(E)
                            2
                          dE

(C18) KILL(LL)$

(C19) DETAYLOR:TAYLOR(DE,E,0,6);
                      2           2                      2            2
(D19)/T/ 2 LL  - LL  A  + (- LL  A  + 6 LL ) E + (- LL  A  + 12 LL ) E
             2     0             1          3           2           4
```

```
           2            3            2            4
+ (- LL   A + 20 LL  ) E  + (- LL   A + 30 LL  ) E
       3           5             4           6

           2            5            2            6
+ (- LL   A + 42 LL  ) E  + (- LL   A + 56 LL  ) E  + . . .
       5           7             6           8

(C20) SOLVEALL(DETAYLOR,E,LL,7)$

(C27) TAYLORSOL: TAYLOR(H(E),E,0,7);
                         2     2     2     3     4     4     4     5
                        A LL  E     A LL  E     A LL  E     A LL  E
                            0           1           0           1
(D27)/T/ LL   + LL   E + --------- + --------- + --------- + ---------
           0     1         2           6          24          120

                                             6     6     6     7
                                            A LL  E     A LL  E
                                                0           1
                                         + --------- + --------- + . . .
                                              720        5040
```

To check this result by "automatic" means, we use MACSYMA's ODE solver, which uses standard textbook recipes, mostly drawn from reference 11. These procedures solve many classes of first and second order linear ordinary differential equations. Anticipating a query about the value of "A", we specify A > 0 below. The answer is reformatted by simplification via RADCAN, and K1, K2 arbitrary constants are related to $LL_0$ and $LL_1$ arbitrary constants by the simultaneous solution of the two linear equations for initial conditions. The result is expanded as a Taylor series to order 7 in E, where it is seen in line D32 to agree to that order with TAYLORSOL generated earlier.

```
(C28) ASSUME(A>0)$

(C29) RADCAN(ODE2(DE,H(E),E));

DERIVD FASL DSK MAXOUT being loaded
loading done
                                - A E          2 A E
(D29)             H(E) = %E         (K2 + %E         K1)
(C30) /* IMPOSE INITIAL CONDITIONS H(0)=LL[0], H'(0)=LL[1] */

IC2(%,E=0,H(E)=LL[0],'DIFF(H(E),E)=LL[1]);
```

```
                                        2 A E
                          (LL  A + LL ) %E         LL  A - LL
                 - A E       0     1                 0      1
(D31) H(E) = %E        (--------------------- + -----------)
                                    2 A                2 A
(C32) TAYLOR(RHS(%),E,0,7)-TAYLORSOL;
(D32)/T/                        0 + . . .
```

We will return briefly to this example in the next section when steps (C19) through (C27) are mathematically reformulated and simplified for the special case of a regular solution to a second order linear differential equation, expanded at the origin.

6. An Introduction to Asymptotic Series

This section will necessarily be very sketchy since asymptotic series are both complicated, and discussed in great detail elsewhere. (see (ref. 2) for example).

Consider the function sin(1/x). It is not possible to construct a Taylor series in ascending powers of x, since there are no derivatives at x=0. The fact that there is an essential singularity at zero is a sufficient barrier to power series expansion. However, for sufficiently large x, when 1/x is sufficiently small, sin(1/x) behaves like 1/x (sin(y) = y + ...).

The notion of an asymptotic series is quite useful in the approximation of functions. Whether or not the series converges is not necessarily important: just as we were willing to deal with a truncated power series, we can deal with a truncated asymptotic series. MACSYMA is capable of producing some series from defining expressions as illustrated below.

```
(C1) TAYLOR(SIN(1/X),X,0,5);
Essential singularity encountered in

                                    1
                                SIN(-)
                                    X

(C2) TAYLOR(SIN(1/X),[X,0,5,ASYMP]);
                         1   1      1
(D2)/T/                  - - ---- + ------ + . . .
                         X      3        5
                             6 X    120 X
```

Unfortunately, many of the most useful asymptotic expansions do not have such a simple structure. For example, instead of a series in descending powers of x, we may need a series in powers of exp(x). A series which MACSYMA cannot "automatically" handle is easily produced via the program given below. The reference to Olver is ref. 2 in the References. We do not define "irregular singularity" or "rank", but the interested reader may refer to ref. 2 for background. Incidentally, this program is a demonstration of the brevity possible in MACSYMA programs for non-trivial mathematical transformations.

```
(C3) /* EXTENSION OF LG (WKBJ)  APPROXIMATION FOR LINEAR 2ND ORDER
      ODE'S IN THE NEIGHBORHOOD OF AN IRREGULAR SINGULARITY
      (SUBCASE:  UNIT RANK AT INFINITY).  SECTION 7.1 IN OLVER.
  (SOLVES
                 W''+F(Z)*W'+G(Z)*W = 0
 GIVING SPECIFIED NUMBER OF TERMS.)
   */

ODE701(FF,GG,WW,Z,TMS):=   /* 701 indicates section 7.1 in Olver */
BLOCK([RHO],
LOCAL(F,G,LAMBDA,MU,A,W),
        /*F[I] and G[I] represent terms in expansion of arguments
        FF and GG */
        F[I]:=LIMIT(Z^I*(FF-SUM(F[J]/Z^J,J,0,I-1)),Z,INF),
        G[I]:=LIMIT(Z^I*(GG-SUM(G[J]/Z^J,J,0,I-1)),Z,INF),
        RHO:(1/4*F[0]^2-G[0])^(1/2),
        IF RHO=0 THEN RETURN(ODE70103()),
           /* SPECIAL CASE OF SECTION 7.1.3 */
        /* lambda[0] and lambda[1] correspond to two
          solutions in series.  Same for mu[0], mu[1]. */
        LAMBDA[I]:=-1/2*F[0]+(-1)^I*RHO,
        MU[I]:=-(F[1]*LAMBDA[I]+G[1])/(F[0]+2*LAMBDA[I]),
        A[0,0]:'K1,  A[0,1]:'K2,    /* arbitrary constants */
        A[S,I]:= 1/ (S*(F[0]+2*LAMBDA[I]))*
           (SUM((LAMBDA[I]*F[J+1]+G[J+1]-(S-J-MU[I])*F[J])*A[S-J,I],
                J,1,S)
           +(S-MU[I])*(S-1-MU[I])*A[S-1,I]),
        W[I]:=%E^(LAMBDA[I]*Z)*Z^MU[I]*SUM(A[S,I]/Z^S,S,0,TMS),
RETURN(WW=W[1]+W[0]))$

(C4) TESTF:(2*Z^2+2*Z+5)/Z^2$

(C5) TESTG:(2*Z+3)/Z^2$

(C6) ODE701(TESTF,TESTG,W,Z,3);
/* solve z^2*W''+(2*z^2+2*z+5)*w +(2*z+3)*w = 0  */
```

```
          7 K2   45 K2  325 K2        - 2 Z
         (---- + ----- + ------ + K2) %E
          2 Z       2       3
                 8 Z    48 Z
(D6) W = ----------------------------------
                         Z
                                            3 K1   5 K1  25 K1
                                            ---- + ---- - ----- + K1
                                            2 Z       2      3
                                                   8 Z   16 Z
                                           + -----------------------
                                                       Z
```

This section is required for exercise 7.1.2 in Olver, so we proceed to fill in the "blank" in the above program namely program ODE70103.

```
(C7) /* OLVER SECTION 7.1.3 "Transformation of Fabry" */

ODE70103():=
BLOCK([F2,G2,NEWF,NEWG,ANS],
        F2:SUBST(Z^2,Z,FF),
        G2:SUBST(Z^2,Z,GG),
        NEWF:2*Z*F2-2*Z*F[0]-1/Z,
        NEWG:Z^2*(4*G2+F[0]^2-2*F[0]*F2),
        IF 2*G[1]=F[0]*F[1] THEN
           /* REGULAR SINGULARITY AT Z=INF: CONVERGENT POWER SERIES */
           /*Method in Olver, Section 5.4, but expand
           at infinity.  See below for expansion at zero. */
             ANS:ODE504INF(NEWF,NEWG,W,Z,TMS) /* AT INF*/
                    ELSE
             ANS:ODE701(NEWF,NEWG,W,Z,TMS),
RETURN(WW=SUBST(SQRT(Z),Z,RHS(ANS)*%E^(-F[0]*Z/2))))$

(C8) TESTF:2/Z$

(C9) TESTG:-(1/4+5/16/Z)/Z$

(C10) /*OLVER EXERCISE 7.1.2 */

ODE701(TESTF,TESTG,W,Z,4);
```

```
                     K1       SQRT(Z)      K2              - SQRT(Z)
            (K1 - -------) %E          (------- + K2) %E
                  SQRT(Z)               SQRT(Z)
(D10)   W = ------------------------- + -------------------------
                       3/4                         3/4
                      Z                           Z

(C11) /* THIS ANSWER HAPPENS TO BE EXACT.  PROOF? BELOW: */

'DIFF(W,Z,2)+TESTF*'DIFF(W,Z)+TESTG*W,%,DIFF,EXPAND;
(D11)                               0
```

Another standard technique for series expansion is the method of Frobenius. Here we dispense only with the case of roots of the indicial equation which do not differ by an integer (or zero). The latter case requires separate, but fairly simple treatment. One example is worked on lines (C13)-(C15).

```
(C12) /* OLVER SECTION 5.4.1:  REGULAR SINGULARITY.
ASSUME WITHOUT LOSS OF GENERALITY EXPANSION AT ORIGIN
(METHOD OF FROBENIUS).  */

ODE504(FF,GG,WW,Z,TMS):=   /*Olver section 5.4.1 */
BLOCK([DISCR,SD],
LOCAL(ALPHA,F,G,A,Q,W),
        F[I]:=LIMIT((Z*FF-SUM(F[J]*Z^J,J,0,I-1))/Z^I,Z,0),
        G[I]:=LIMIT((Z^2*GG-SUM(G[J]*Z^J,J,0,I-1))/Z^I,Z,0),
        DISCR:(F[0]-1)^2-4*G[0],   /* DISCRIMINANT OF INDICIAL EQUATION */
        SD:RADCAN(SQRT(DISCR)),
        ALPHA[I]:=(-F[0]+1+(-1)^I*SD)/2,  /* QUADRATIC SOL. */
        Q(X):=X*(X-1)+F[0]*X+G[0],
        W[I]:=Z^ALPHA[I]*SUM(A[S,I]*Z^S,S,0,TMS),
        A[S,I]:=-SUM(((ALPHA[I]+J)*F[S-J]+G[S-J])*A[J,I],J,0,S-1)/
                Q(ALPHA[I]+S),
        A[0,0]:'K1,A[0,1]:'K2,    /*ARBITRARY CONSTS */
        IF INTEGERP(ALPHA[0]-ALPHA[1]) THEN ODE50501()  /*  ROOTS DIFFER
              BY INTEGER OR 0 */
  ELSE RETURN(WW= W[0]+W[1]))$

(C13) FF:%E^Z/Z$
(C14) GG:-3*COS(Z)/Z^2$
(C15) RATSIMP(ODE504(FF,GG,W,Z,3));
             2 SQRT(3)                             3
(D15) W = (Z           ((176 SQRT(3) - 253) K1 Z
                                    2
 + (108 - 117 SQRT(3)) K1 Z  + (72 SQRT(3) - 432) K1 Z + 792 K1)

                                     3                        2
 + (- 176 SQRT(3) - 253) K2 Z  + (117 SQRT(3) + 108) K2 Z

                                                  SQRT(3)
 + (- 72 SQRT(3) - 432) K2 Z + 792 K2)/(792 Z        )
```

To close this section, we show how to generate, in a rather brief program, a Taylor series expansion we have seen before; the solution to DIFF(H(E),E,2)-$A^2$*H(E)=0.

```
(C16) /* EXPANSION IN SERIES, ORDINARY POINT.
ASSUME WITHOUT LOSS OF GENERALITY EXPANSION AT ORIGIN
OLVER SECTION 5.3.2 */

TAYSER(FF,GG,WW,Z,TMS):=
BLOCK([],
LOCAL(A,F,G),
        A[0]:'K1,A[1]:'K2,
        A[S]:=-1/S/(S-1)*SUM(F[J]*(S-J-1)*A[S-J-1]+G[J]*A[S-J-2],J,0,S-2),
        F[I]:=LIMIT((FF-SUM(F[J]*Z^J,J,0,I-1))/Z^I,Z,0),
        G[I]:=LIMIT((GG-SUM(G[J]*Z^J,J,0,I-1))/Z^I,Z,0),
RETURN(WW=SUM(A[S]*Z^S,S,0,TMS)))$


(C18) TAYSER(0,-A^2,W,E,7);
           6  7       4  5       2  3              6  6       4  4
          A  E  K2   A  E  K2   A  E  K2          A  E  K1   A  E  K1
(D18) W = -------- + -------- + -------- + E K2 + -------- + --------
            5040       120         6                 720         24

                                                                2  2
                                                               A  E  K1
                                                           + -------- + K1
                                                                  2
```

This is the same as D27 of the previous section.

## 7. The Use of Newton Iteration over a Power Series Domain

A powerful technique for solving algebraic problems is pointed out in references 8 and 9. We restate Lipson's theorem 3.1 (ref. 9) to justify the following constructions.

**THEOREM**: Let f(x) be a polynomial with coefficients in a power series domain (series in t with coefficients in F) D=F[[t]]. Let a in F be an O(t) approximation to a root of f(x) (i.e. x=a is a solution to f(x)=0 when t=0). Furthermore, suppose that a satisfies $f'(a) \neq 0$ when t=0 (where the prime indicates differentiation with respect to x)

Then the sequence of iterations $x_0=a$, $x_1$, ... computed according to

$$x_k = (x_{k-1} - f(x_{k-1})/f'(x_{k-1})) \bmod t^{(2^k)}$$

is such that $x_k$ is an $O(t^{(2^k)})$ approximation to x.

Reference 8 generalizes this result somewhat by explaining how an iteration can be constructed for a polynomial f(x) which does not satisfy the condition on f'(x). This "Newton polygon" calculation will not be demonstrated here.

We note in passing that our earlier examples do not satisfy the requirements of this theorem.

The following protocol does not demonstrate the most efficient formulation of this iteration, since one can concoct (as demonstrated in ref. 9) efficient Horner's rule evaluation of a polynomial and its first derivative at a power-series point, and furthermore, the essential computations can be done by asymptotically faster methods (ref. 7). Yet, since one is much more likely to be interested in the first few terms of an expansion than any others, an $O(n^2)$ or slightly worse algorithm for n-terms is not objectionable.

```
(C1) /* NEWTON´S METHOD FOR ROOT-FINDING OVER A POWER SERIES
DOMAIN */

/* INPUTS:
EX= EXPRESSION IN VARIABLES W AND T.  EX=0 WILL BE SOLVED
   APPROXIMATELY
   FOR W(T) TO ORDER N OR HIGHER.
AROOT IS A ZERO OF EX WHEN T=0, SUCH THAT DIFF(EX,W) WITH T=0
   IS NONZERO.  (THIS CONDITION IS CHECKED.) */

NEWTONROOT(EX,W,T,N,AROOT):=
BLOCK([DEX,S,I],
DEX:DIFF(EX,W),
/* CHECK INITIAL CONDITIONS IN NEXT  IF STATEMENT */
IF TAYLOR(SUBST(AROOT,W,EX),T,0,0)#0
   OR TAYLOR(SUBST(AROOT,W,DEX),T,0,0)=0
   THEN RETURN(PRINT("NOT ABLE TO EXPAND AT ", AROOT)),
S:AROOT,
FOR I:1 NEXT 2*I+1 WHILE I<=N DO
  (S:RATDISREP(SUBST(S,W,S-TAYLOR(EX/DEX,T,0,I))),
   S:TAYLOR(S,T,0,2*I+1) /* PREPARE FOR NEXT ITERATION */ ),
RETURN(S))$

(C2) /* THE FOLLOWING EXAMPLES ARE TAKEN FROM REF. 9. */

/* PROBLEM 1.  COMPUTE A SQUARE ROOT OF A=1+T+2*T^2+3*T^3+...
TO ORDER 8 ERROR. */

NEWTONROOT(X^2-(1+T+2*T^2+3*T^3+4*T^4+5*T^5+6*T^6+7*T^7),X,T,7
,1);
                         2        3         4         5         6         7
               T    7 T     17 T     139 T     263 T     995 T    1969 T
(D2)/T/ 1 + - + ---- + ----- + ------ + ------ + ------ + -------
               2     8       16       128       256       1024      2048

                                                                     + . . .

(C3) /*PROBLEM 2.  COMPUTE A SOLUTION TO A CUBIC */

NEWTONROOT(X^3-2/(1-T)*X+1,X,T,7,1);
                            2        3        4          5           6
(D3)/T/ 1 + 2 T - 6 T  + 58 T  - 622 T  + 7506 T  - 96822 T

                                                                 7
                                                   + 1307466 T  + . . .

(C4) /*PROBLEM 3. REVERT T=ATAN(X) TO FIND A SERIES FOR TAN(T)
 */

NEWTONROOT(ATAN(X)-T,X,T,7,0);
                                3      5       7
                               T    2 T    17 T
(D4)/T/                  T + -- + ---- + ----- + . . .
                               3     15     315
```

## 8. Comments on the Implementation

Several notational problems seem apparent. If a TPS is displayed as $Y = 1 + x + \ldots$, does this mean that $Y - 1 - x$ is $O(x^2)$? How would the display differ if the difference was $O(x^3)$? The ellipsis is insufficient, and $1 + x + O(x^2)$, if such is the case, would resolve the question. This information is available internally, in most cases, anyway. As pointed out by R. Zippel (private communication), how can one compute n in $\sin(x)^2+\cos(x)^2-1 = 0 + O(x^n)$ ? A calculus of orders seems to be the next step in this direction: $(1+O(x^2))*(1+O(y)) = 1 + O(x^2*y)$, not $1 + \ldots$. Addition and other operations would have to be implemented, along with a careful treatment of the asymmetrical use of this notation on the left and right hand sides of equations.

Another deficiency, not illustrated in this paper exists in terms of the consistency of TPS operations in parts of MACSYMA. For example, matrix operations with TPS entries forces a conversion to a non-TPS form. In the process, information is lost which can be of considerable benefit. It also appears that significant time savings may be possible by recognition of TPS matrices as a special case: computing the inverse of a matrix of TPS entries can be done in a variety of ways by matrix-wise series expansion, for example.

The implementation of infinite summations "SUM"s is currently in flux, because of important work due to R.W. Gosper (reported in this Proceedings). While it is possible to solve the equation (C1) as mentioned earlier, to get a closed-form formula for $LL_n$ in finite terms, the manipulation is not yet routine using MACSYMA.

What is needed, minimally, is the capability of moving independent variables both in and out of summations: $a*\mathrm{sum}(x^i,i,0,inf) <==> \mathrm{sum}(a*x^i,i,0,inf)$, changing the index: $x^a*\mathrm{sum}(x^i,i,0,inf) <==> \mathrm{sum}(x^{i+a},i,0,inf) <==> \mathrm{sum}(x^i,i,a,inf)$, taking terms out of sums: $\mathrm{sum}(a_i,i,0,inf) ==> a_0+\mathrm{sum}(a_i,i,1,inf)$ or $\mathrm{sum}(a_i,i,0,n) ==> \mathrm{sum}(a_i,i,0,n-1) + a_n$. Suitable generalizations of these transformations, plus a neat methodology for specifying which transformation to use where would provide a basic facility. More elaborate simplifications can be programmed, but without this type of facility, the lone user has a difficult time. We note that thisproposed facility is different from one which does exist in MACSYMA, namely the simplification of sums to closed forms when possible, mentioned in the previous paragraph.

## 9. Conclusions

We hope we have given a sufficient number both of main-line and incidental comments concerning the use of series, especially in MACSYMA, to illustrate the principal well-understood approaches. While the details of derivation of these methods, and the underlying (sometimes quite sophisticated) programming and mathematical algorithms have not been explained in this paper, sufficient information on these topics is available in the references.

We have deliberately avoided discussion of methods for convergent or asymptotic series approximations of integral equations, and transcendental equations. This is not because of lack of material: rather, there is a wealth of material, especially on integral approximation and integral equation solution. The work of Stoutemyer (ref. 13) originally in REDUCE has been made available in MACSYMA by Richard Bogen (reported in these Proceedings). Early work by Paul Wang (ref. 14) and Seth Chaiken at MIT provide procedures for integral approximation by the methods of stationary phase, steepest descent, and other schemes. There are a growing number of references to work in other systems, principally REDUCE, FORMAC (ref. 15) and ALTRAN along the lines of the more straightforward rational methods. These may be identified through recent ACM SIGSAM Bulletin listings of abstracts. We would like to note the interesting use of Taylor series in a combined numerical/symbolic mode as in (refs. 16, 17). The idea in these papers is to use symbolic methods in a compiler as a technique for producing numerical approximation programs. By separating the two passes, machine resources can be optimized for the differing requirements of symbolic and numerical routines.

We hope to classify, describe, and extend approximation work in a variety of areas, including but not limited to the areas explored in this paper, at a later time. A number of researchers have examined simple applications of the method of successive approximation (Picard's method) in a symbolic context. The combination of this technique with power series is very promising.

A common question raised by the automatic solution of equations by series is: How do we know these methods produce a convergent series, or how can we find the radius of convergence. The answer to both of these questions is: we use the same methods that mathematicians use by "hand"; there is very little magic in the automation of these methods. They are for the most part "formal" methods whose convergence can be guaranteed only by additional consideration of the problem at hand. Indeed, some of the asymptotic methods will usually produce a divergent series; this does not mean the result is meaningless or useless, since such series have a wide use in the literature.

Significantly absent from this paper is a discussion of the validity of series solutions, and how to diagnose the appropriateness of various approaches to solving algebraic or differential equations by approximation. This problem is probably best solved by practitioners in each given area who are familiar with particular approaches relevant in their special problem domains. The tools provided by MACSYMA, plus simple programs as outlined above serve as early steps toward more useful cooperation between the applied mathematician and the computer.

## REFERENCES

1. MACSYMA Users Manual. Lab. for Computer Sci., Massachusetts Inst. Technol., 1975.

2. Olver, F. W. J.: Asymptotics and Special Functions. Academic Press, 1974.

3. Wasow, W.: Asymptotic Expansions for Ordinary Differential Equations. John Wiley and Sons (Interscience), 1965.

4. Zippel, R. E.: Univariate Power Series Expansions in Algebraic Manipulation. Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation. Aug. 1976, pp. 198-208.

5. Norman, A. C.: Computing with Formal Power Series. ACM Trans. on Math Software, vol. 1, no. 4, Dec. 1975, pp. 346-356.

6. Brown, W. S., ALTRAN User Manual. Bell Telephone Lab Inc., 1973.

7. Brent, R.; and Kung, H. T.: Fast Algorithms for Manipulating Formal Power Series. Tech. Rep., Computer Sci. Dep., Carnegie-Mellon Univ., Jan. 1976.

8. Kung, H. T.; and Traub, J. F.: All Algebraic Functions Can be Computed Fast. Tech. Rep., Computer Sci. Dep., Carnegie-Mellon Univ., July 1976. July, 1976.

9. Lipson, J.D.: Newton's Method: A Great Algebraic Algorithm. Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation, Aug. 1976, pp260-270.

10. Knuth, D. E., The Art of Computer Programming, Volume 2: Seminumerical Algorithms. Addison-Wesley Pub. Co., Inc., 1969.

11. Boyce, W. E.; an d DiPrima, R. C.: Elementary Differential Equations. JohnWiley & Sons, Inc., 1969.

12. Nayfeh, A. H.: Perturbation Methods. John Wiley & Sons, Inc., 1973.

13. Stoutemyer, D. R., Analytical Solution of Integral Equations Using Computer Algebra. UCP-34, Computer Sci. Dep., Univ. of Utah, June 1975. (to appear, ACM Trans on Math. Software).

14. Wang, P. S.: Application of MACSYMA to an Asymptotic Expansion Problem. Proceedings of the 27th ACM Annual Conference- Volume 2, Aug. 1972, pp. 844-850.

15. Hanson, J. N.: Experiments with Equation Solutions by Functional Analysis Algorithms and Formula Manipulation. J. Computer Physics, vol. 9, 1972. pp. 26-52.

16. van de Riet, R. P.: The Automatic Solution of Partial Differential Equations by Means of

Taylor Series Using Formula-Manipulation Methods. ACM SIGSAM Bulletin, no. 28. Dec. 1973, pp. 33-36.

17. Barton, D.; Willers, I. M.; and Zahar, R. V. M.: Taylor Series Methods for Ordinary Differential Equations - An Evaluation. Mathematical Software. J. Rice, ed., Academic Press, 1971, pp. 369-390.

34

# POWER SERIES SOLUTIONS OF ORDINARY DIFFERENTIAL EQUATIONS IN MACSYMA*

Edward L. Lafferty
The MITRE Corporation

## INTRODUCTION

A program has been developed which extends the differential equation solving capability of MACSYMA to power series solutions and is available via the SHARE library. The program is directed toward those classes of equations with variable coefficients (in particular, those with singularities) and uses the method of Frobenius. Probably the most important distinction between this package and others currently available or being developed is that, wherever possible, this program will attempt to provide a "complete" solution to the equation rather than an approximation, i.e., a finite number of terms. This solution will take the form of a sum of infinite series.

The Frobenius method stated simply here as a refresher (see Ref. 1, p.189 for a more complete treatment) asserts that for a homogeneous, linear, differential equation of the form:

$$y'' + P(x)\, y' + Q(x)\, y = 0 \qquad (1)$$

where P and Q are polynomials in X, then at the ordinary point, X=Xp, a solution exists of the form:

$$Y = \sum_{N=0}^{INF} A_N X^N \qquad (2)$$

where $A_0$ and $A_1$ are arbitrary constants and are the values of Y(0) and Y'(0).

The method further asserts that for a regular singular point, X=Xs, the solution is the sum of two linearly independent solutions:

---

*. The work described in this paper was begun by B. Kuipers in 1973 and the author is indebted to him for several ideas and at least one routine. In addition, the author wishes to acknowledge the encouragement and assistance of J. P. Golden throughout the course of the effort.

$$Y = K2 \left( \sum_{N=0}^{INF} B_N (R2)\, X^{R2 + N} \right) + K1 \sum_{N=0}^{INF} A_N (R1)\, X^{R1 + N} \qquad (3)$$

where r1 and r2 are the exponents of the singularity.

There are two special cases:

i) r1=r2, in which the B's are found to be A' (r1)* and the second solution contains a logarithmic term;

and ii) r1-r2=S, an integer, in which the B's are found to be (r-r2) A'(r2) and the second solution contains a logarithmic term except for the very special case in which it is found that some one of the A's (in addition to $A_0$) is arbitrary (see ref. 2 for a particularly complete treatment of this case).

At top level, after a LOADFILE(SERIES,FASL,DSK,SHARE), the program is called by the statement SERIES(equation, y, x), where "equation" is a second order linear ordinary differential equation and "y" and "x" are the dependent and independent variables respectively. (Of course, the dependencies between the variables must be established prior to typing the equation.)

## RATIONALE FOR COMPLETE SERIES SOLUTIONS

Virtually all elementary courses in differential equations introduce the student to the power series method at an early stage, and many such courses continue to solve problems by using direct substitution of the power series into the equation and determining the recurrence relation. Even in those instances where the student is introduced to approximation methods using Taylor coefficients to determine a recurrence relation for each term in the solution, the authors (for example, see references 1,2,3,4,5) frequently will revert to direct substitution so that the student may better understand the behavior of the variables, arbitrary constants, and parameters of the equations.

While the mathematician who is intimately familiar with the theory and practice of solving differential equations may have no difficulty recognizing instantly that certain forms are Bessel equations, Legendre equations, or hypergeometrics, the average mathematician or, more importantly, the engineer who has only a superficial understanding of this subject may not. Early in the pursuance of this project, I confronted several advanced degreed mathematicians with the equation (later found in Ref. 6, p. 97):

---

* A'(r1) denotes the partial derivative of A with respect to R evaluated at r1.

$$\frac{d^2Y}{dX^2} - \frac{\frac{dY}{dX}}{X} + Y = 0 \qquad (4)$$

Only one of five even offered a tentative identification of this equation as a Bessel, and of the five, two proceeded to solve it by the method of Frobenius. (The above equation will also be used throughout this paper to illustrate some of the internals of the program. These results will be numbered (4a), (4b), etc.)

Summarizing, then, the reason for including such a capability within MACSYMA, we find it useful for:

a. the student who wishes to understand the theory;

b. the mathematician or engineer who may fail to recognize the particular form; and

c. the theoretical mathematician working on advanced forms who prefers to start from basic principles.

## THE METHOD

The first step in the solution process is the diagnosis of the equation for singularities. While only one (the one at which a solution is desired) of these singularities is of concern to the program for what is to follow, it may be generally useful for the user to know at what points the equation possesses poles of one sort or another. The program uses Stoutemyer's ZEROSANDSINGULARITIES routine because of its generality, e.g., it will find poles of log(x), tan(x), etc., as well as polynomials.

The indicial equation is computed from:

$$R^2 + (P_0 - 1) R + Q_0 = 0 \qquad (5)$$

where $P_0$ and $Q_0$ are the values of $P(x)$ and $Q(x)$ at $x=0$, computed by:

$$P(0) = \lim_{x \Rightarrow 0} (xP(x)) \quad \text{and} \quad Q(0) = \lim_{x \Rightarrow 0} (x^2 (Q(x)) \qquad (6a,6b)$$

Solving this equation for R yields the roots r1 and r2. From these roots, it is determined whether the final solution will contain a logarithmic term, i.e., if r1 = r2 or if r1 - r2 = S (an integer). In addition, the very special case is detected in which the roots differ by an integer, but the solution does not possess a logarithmic term, i.e., wherein:

$$A_{i-1} = 0$$

and therefore, the coefficient $A_i$ is finite and arbitrary.

At this point it is time to begin the direct substitution of the series:

$$Y = \sum_{N=0}^{INF} A_N X^{N+R} \qquad (7)$$

(r=0 for an ordinary point) into the equation and evaluate the derivatives. The MACSYMA POWERSERIES function is then used to determine a single series for the entire left-hand side of the equation.*For our example (eq. 4) the result is:

$$\sum_{I6=0}^{INF} A_{I6} X^{R+I6} + (A_{I6} R^2 + (2\ I6 - 2)\ A_{I6} R \qquad (4a)$$

$$+ (I6^2 - 2\ I6)\ A\ )\ X^{R + I6 - 2}$$

---

* While this is a form of overkill for this operation, the routine can handle the job and will be necessary for later operations.

An important feature of the program is the routine which computes the recurrence relation. This is done by removing the summation sign and by equating like powers of the independent variable to zero. Some program shortcuts are taken in this process, but it is essentially a replication of what is done by hand. The example result looks like this:

$$(A_N R^2 + 2 N A_N R - 2 A_N R + N^2 A_N - 2 N A_N + A_{N-2}) X^{R+N-2} \tag{4b}$$

The recurrence relation which is available to the user is expressed in the form:

$$A_N = f(N) A_{N-M} \tag{8}$$

or for singular points:

$$A(R)_N = f(N,R) A_{N-M} \tag{8a}$$

if M>1, there are some adjustments which must now be made to the rest of the solution.* For our equation the recurrence relation becomes:

$$A_N = - \frac{A_{N-2}}{(R + N - 2)\ (R + N)} \tag{4c}$$

In some cases this is as far as the user may want to go in determining the solution to his equation. In particular, if the function on the right-hand side of equation (8) contains more than one "A" term, no easy simplification method is available, and the program can then compute only a finite number of terms for each of the solutions to be determined. In the special cases, of course, appropriate differentiation of the recurrence relation is required before this can be done for the second solution. In all cases where the recurrence relation is expressed as a single term in A, the program then proceeds to determine a complete solution as an infinite series. It is here that the system must perform two interesting functions which will be described in the next section, i.e., differentiation of partial products and the simplification of partial products[**] into factorials and polynomials.

---

* These will not be described in detail here for lack of space, but suffice it to say that some "A" values must be set to zero in the solution, and the exponents of the independent variable must be adjusted to reflect the missing terms.

The special cases $r1 = r2$ and $r1 - r2 = S$ are handled by the following relation:

$$B_N = \left. \frac{d\,A(R)_N}{dR} \right| \text{evaluated at } r1 \tag{9}$$

and the solutions will have the logarithmic form. For our equation which is the $r1 - r2 = S = 2$ case, the final solution would be:

$$Y = B_0 \left( - \frac{\displaystyle\sum_{K=1}^{INF} \frac{(HARM(1, K) + HARM(1, K - 1)) (- 1)^K X^{2K}}{2^{2K} (K - 1)!\ K!}}{2} + LOG(X) \left( \sum_{K=1}^{INF} \frac{(- 1)^K X^{2K}}{2^{2K} (K - 1)!\ K!} \right) + \frac{1}{2} \right) + A_0 \sum_{K=1}^{INF} \frac{(- 1)^K X^{2K}}{2^{2K} (K - 1)!\ K!} \tag{4d}$$

[*]

## THE COMPLETE SOLUTION

Probably the most interesting section of the program is that which performs the transformation from equations (4c) to (4d). This involves expressing the recurrence relation as an infinite series of partial products in N, the index, and R, the general exponent of the singularity. This must be differentiated with respect to R in the two special cases (eq. 9) and then simplified.

---

[**] The term "partial products" is used to distinguish them from completely finite products, i.e., those that can be computed by the function, PRODUCT, and infinite products. Another commonly accepted term is "indefinite" products.

[*] The HARM function and its product analog, FFF, is discussed in the next section.

Two slightly different approaches were taken to this problem and code for both currently exist. The first, retaining the PRODUCT and SUM forms throughout, is deemed to be inferior and will not be described here; but a package does exist which can handle the simpler equations using this technique.

In working with the more complicated cases, it was found useful to change the representation of the partial products to the more compact "factorial function" (FFF) of Rainville (see Ref. 3, pp 109-112).

$$FFF(exp,n)=exp(exp+1)(exp+2)\ldots(exp+n-1) \qquad n \geq 1 \tag{10}$$

and

$$FFF(exp,0)=1, \; exp \neq 0 \tag{11}$$

and the familiar special case:

$$FFF(1,n)=n! \tag{11a}$$

This method has a distinct advantage in that quotients of FFF's simplify easily and the gradient of FFF with respect to a variable first argument is simply

$$\frac{d(FFF(exp,n))}{dr} = HARM(exp,n) \; FFF(exp,n) \tag{12}$$

where HARM(exp,n) is the partial sum of the harmonic series:

$$HARM(exp,n) = \sum_{k=1}^{n} \frac{1}{exp + k} \tag{13}$$

and the special case:

$$HARM(1,n) = SUM(1/k,k,1,n) \tag{13a}$$

Simplification of factorial function quotients is accomplished using the following algorithm:

```
                 FFF(alph,nalph)/FFF(bet,nbet):=

                      FFF(m+1,n-m)^(pow)
          ----------------------------------------               (14)
          FFF(min(alph,bet),abs(rho))^signum(rho)
```

where:

```
                 rho=alph-bet
                 pow=polysign(alph+nalph-bet-nbet)
                 m=min(alph+nalph-1,bet+nbet-1)
         and     n=max(alph+nalph,bet+nbet)-min(alph+nalph,bet+nbet)
```

thus giving nicer looking results. More importantly it allows the easy removal of the troublesome denominators (see Ref. 3, p.44) which occur in case ii) above since

```
                          FFF(r-r2,1)
                         -------------                           (14a)
                         FFF(r+k-r2,n)
```

simplifies by the above algorithm to

```
                                1
                  ------------------------------                 (14b)
                  FFF(r-r2-1,1) FFF(r-r2+1,n-1)
```

In addition, the compact notation for FFF and HARM may lead eventually to automatic recognition of closed forms by MACSYMA, or at least assist a user's visual recognition process.

## USER OPTIONS

There are several facilities which the user may control. In particular, he may control the point around which the solution is determined by setting the variable POINTEXPAND: [0] and the maximum number of terms to be computed in a finite series by setting the variable NUMTERMS: [5]. The above variables have only limited use in the program currently. However, they have ultimately a more general use. In particular, the POINTEXPAND flag is used to determine whether the equation being processed has singularities at that point. However, if the variable is not zero, the translation will not be made to the new point and, therefore, although the diagnosis will be correct, the solution will not. NUMTERMS is used for computing a partial series as well as for computing the Taylor coefficient of polynomials P and Q and may be useful in those cases where a complete solution is not possible.

In addition to the above options, the user may set the flag VERBOSE1 [FALSE] to TRUE to obtain automatic printout of the diagnostic information relating to the equation, i.e., the recurrence relation, the location and type of singularities, and the roots of the indicial equation. This may be particularly useful if the routine is attempting to solve an equation for which it is not now equipped, i.e., irregular singular points, complex roots, equations of order higher than two, etc., or where the user is only interested in the diagnostics rather than the complete solution.

## THE FUTURE

In order to produce a program in a reasonable period of time, certain restrictions were imposed which can, with varying amounts of difficulty, be relaxed, and there are some basic extensions which might prove valuable in the future. We will attempt to enumerate some of these here. It should be noted that several of the internal routines were coded with these extensions in mind, i.e., certain data are now computed which are not used in the current program, and these will be noted where applicable.

### Higher Order Equations

The method of Frobenius readily extends to higher order linear differential equations and up to the point of diagnosis, this has been generalized. This, in the author's opinion, is the most valuable future improvement which might be undertaken. It is required that the n roots of an nth order equation be computed, n arbitrary constants be allocated, and n solutions be generated. Even the special cases of r1 = r2 ... = rn and r1 - r2, r2 - r3, ..., rn-1 - rn = S can be solved by taking n derivatives of the recurrence relation, although this may require some thought (see Ref. 3, p. 120).

### Solution Around Points Other Than Zero

While the user can easily transform his equation into one whose solution can be determined around zero by the transformation:

$$\mathrm{newX}=\mathrm{X}-\mathrm{point} \tag{15}$$

it would be a trivial matter for the program to recognize POINTEXPAND = 0 and perform the translation and retranslation for him.

### Complex Roots

An unnecessary restriction exists in the current program for r1, r2 complex. The restriction can be relaxed rather easily by computing the real parts of r1 and r2 and using them in the diagnosis and solution of the equation as follows:

$$RE(r1)>RE(r2) \tag{16}$$

## Irregular Singularities

At present the program will not attempt a solution around an irregular singular point. It may be possible to attempt complete solutions to the equation around an irregular singularity, but some work must be done to determine the validity of such solutions (see Ref. 3, p. 136). There are, however, other approximation methods for these cases which may be adequate in view of the work involved to incorporate an extension to the program.

## Convergence Tests

A useful feature could be added to the program at the point of generation of the recurrence relation or after completion of the final solution which would perform a test for convergence. This would give the user important additional information regarding the radius of convergence and validity of the solutions thus obtained.

## User Cueing

It was assumed in the construction of this capability that the user could substitute the values for arbitrary constants after the solution was obtained. For certain applications, it might be desirable for the program to interrupt its execution to ask the user for the initial values of the dependent variable and its derivatives. In addition, where variable parameters are used instead of constants in the polynomial coefficients, P and Q, the program does not currently make assumptions regarding the ranges and will, for example, produce solutions in terms of MIN (parameter, 0 ) and MAX (parameter, 0). The user may, of course, reenter the routine having made assumptions about the parameters. (See the final example of this paper.) However, since these relationships could, in fact, cause a major variation in the solution type, it would be desirable for the program to sense these ambiguities qnd cue the user for his assumptions prior to final diagnosis of the equation and initiation of the solution.

## Non-Homogeneous Cases

At present the program solves only homogeneous linear differential equations of the form:

$$Y''+P(x)Y'+Q(x)Y=0 \tag{17}$$

Another particular solution may be obtained for equations of the form:

$$Y''+P(x)Y'+Q(x)Y=F(x) \tag{17a}$$

provided the function on the right-hand side can be expressed as a power series. Some modification will be required to the program to recognize this case as well as to insure that the routine which computes the recurrence relation does not encounter any problems in combining the additional series.

### Non-Polynomial Coefficients

If the functions P and Q can be expressed in terms of power series, then a modification of the program can be made similar to the non-homogeneous case which would allow solution by this method. Again there must be some work done to determine whether the routines will encounter expressions beyond their capability.

## CONCLUSION

Several more elaborate extensions come to mind, but they require more than a mere modification of this package. The first would be to incorporate this capability into the current ODE solving capability of MACSYMA such that in situations where ODE cannot recognize a particular form, it automatically attempts a power series solution. Naturally, certain tests should be made by ODE (or alternatively, built into the SERIES package) prior to this attempt depending on the current state of its capability.

A final and far more extensive venture which has been suggested by others and is highly endorsed in this paper is the extension of MACSYMA's differential equation solving into the realm of systems of differential equations similar to that currently available for algebraic equations in LINSOLVE, SOLVE, and ALGSYS. This is a project worthy of serious consideration by the community at large and will require the resources of more than a single individual since, in order to do it justice, all of the differential equation capabilities should be examined for possible inclusion in such a system.

## EXAMPLES

The following section contains examples of several of the cases noted above, i.e., solution around:

1. simple ordinary point;
2. ordinary point in which one or both solutions truncate after a finite number of terms;
3. regular singular point (r1 - r2 = S) but the solution does not contain a logarithmic term;
4. the generalized hypergeometric equation in which the user makes an initial assumption.

Note that in the text we have already shown an example of the logarithmic case of 3. above and the reader is directed to the SHARE demo file for a more complete set of examples.

```
(C6) DEPENDENCIES(Y(X));
(D6)                              [Y(X)]

(C7) /*ordinary points*/

EQ1:DIFF(Y,X,2)+3*X*DIFF(Y,X)+3*Y=0;
                            2
                           d Y       dY
(D7)                       --- + 3 X -- + 3 Y = 0
                             2       dX
                           dX
(C8) SERIES(%,Y,X);
                    INF                         INF
                    ====       K  2 K + 1       ====       K  2 K
                    \     (- 3)  X              \     (- 3)  X
(D8)        Y = A  ( >    ---------------) + A   >    -----------
                 1  /          3      K       0 /          K
                    ====   FFF(-, K) 2          ====     2  K!
                    K = 0      2                K = 0
(C9) /*Truncation of a series term*/
 EQ2:(1+X^2)*DIFF(Y,X,2)-2*Y=0;
```

```
                                    2
                            2      d Y
(D9)                       (X  + 1) --- - 2 Y = 0
                                      2
                                    dX
(C10) SERIES(Z,Y,X);
                                   INF
                                   ====            K  2 K + 1
                       2           \           (- 2)   X
(D14)         Y = A  (X  + 1) - A   >    ---------------------
                   0             1 /                         K
                                   ====  (2 K - 1) (2 K + 1) 2
                                   K = 0
(C42) /*roots a positive integer-non-log case*/

EQ8:X*DIFF(Y,X,2)-(4+X)*DIFF(Y,X)+2*Y=0$

(C43) SERIES(%,Y,X);
                 INF
                 ====                 K                    2
                 \                   X                     X    X
(D22) Y = 60 A   ( >    --------------------------) + A  (-- + - + 1)
              5  /      (K - 2) (K - 1) K (K - 5)!     0  12    2
                 ====
                 K = 5


(C25) /*The generalized form of the hypergeometric is:*/
HY1:X*(1-X)*DIFF(Y,X,2)+(GAM-(ALPH+BETA+1)*X)*DIFF(Y,X)-ALPH*BETA*Y=0;
                                                                   2
       dY                                                        d Y
(D26)  -- (GAM - X (BETA + ALPH + 1)) - ALPH Y BETA + (1 - X) X --- = 0
       dX                                                          2
                                                                 dX

(C27) /*since we already know that SERIES will be confused by the paramet
ers*/

ASSUME(1-GAM>0)$

(C28) SERIES(HY1,Y,X);
```

```
(D38) Y = A
           0

  INF
  ====                                  K - GAM + 1
  \     FFF(- GAM + ALPH + 1, K) X               FFF(BETA - GAM + 1, K)
 ( >    -----------------------------------------------------------------)
  /                          FFF(2 - GAM, K) K!
  ====
  K = 0

      INF
      ====               K
      \     FFF(ALPH, K) X  FFF(BETA, K)
 + B   >    ---------------------------
    0 /           FFF(GAM, K) K!
      ====
      K = 0
```

## REFERENCES

1. Boyce, W. E.; and DiPrima, R. C.: Elementary Differential Equations. Second ed., John Wiley & Sons, Inc., 1969.

2. Coddington, E. A.: An Introduction to Ordinary Differential Equations. Prentice-Hall, Inc., 1961, pp. 162-166.

3. Rainville, E. D.: Intermediate Differential Equations. Second ed., Macmillan, 1964.

4. Hildebrand, F. B.: Advanced Calculus for Engineers. Prentice-Hall, Inc., 1949.

5. Rainville, E. D.: Elementary Differential Equations. Macmillan, 1949.

6. Watson, G. N.: Theory of Bessel Functions. Second ed., Cambridge University Press, 1958.

# Radical Simplification Made Easy*

Richard E.B. Zippel
*Laboratory for Computer Science*
*Massachusetts Institute of Technology*

It is a fortunate person who has not been stymied by an algebraic manipulation system which was unable to manipulate fully the algebraic numbers and functions which occured in a problem. Here we see three distinct types of problems. Some simplifiers are "gullible" enough to be coaxed into erroneous sequences of transformations such as:

$$1 = \sqrt{1}\sqrt{1} = \sqrt{1} = \sqrt{(-1)(-1)} = \sqrt{-1}\sqrt{-1} = -1.$$

On the other hand, there are the "conservative" simplifiers which refuse to reduce expressions like $\sqrt{6} - \sqrt{2}\sqrt{3}$ to zero. This conservatism is at least partially justified by the sort of problems into which the gullible simplifier can fall. The third and final deficiency in algebraic simplifiers (and the one which we will dwell on the most) may be described as the problem of the naive simplifier. Typical of these sorts of problems is:

$$\sqrt{5 + 2\sqrt{6}} = \sqrt{2} + \sqrt{3}.$$

Admittedly many users are themselves guilty of being naive in this sense (the above identity is not really obvious), but for some reason we don't seem willing to accept this naiveté on the part of our systems.

Some previous work has been done on the problem of simplification of algebraic expressions. S.L. Kleiman (ref. 1) did early work on the problem in a more general context. Both B.F. Caviness (ref. 2) and R.J. Fateman (ref. 3) did work on unnested radicals in their theses and have written a recent summary of their work (ref. 4). Our work generalizes all the results on simplification of radicals contained in these two theses. For the sake of simplicity all the examples given here are in algebraic number fields. However, the results are fully general and depend upon only the characteristic of ground field.

---

*This work was supported, in part, by the United States Energy Research and Development Administration under Contract Number E(11-1)-3070 and by the National Aeronautics and Space Administration under Grant NSG 1323.

## Basic Definitions

We will need some mathematical terminology in this discussion. If $k$ is a field then the field of rational functions in $\alpha$ over $k$, $K = k(\alpha)$, is called an *extension* of $k$. If $\alpha$ is the zero of some irreducible polynomial with coefficients in $k$, $p(x)$, then $K$ is said to be an algebraic extension of $k$ and $\alpha$ is said to be *algebraic* over $k$. Otherwise $K$ is a *transcendental* extension of $k$ and $\alpha$ is transcendental over $k$. $K$ is a $k$ vector space of finite dimension if and only if $\alpha$ is algebraic. The *degree* of $K$ over $k$, written $[K:k]$ is finite when $\alpha$ is algebraic and in which case is equal to the degree of $p(x)$. If $p(x)$ consists of two terms, i.e. $p(x) = ax^n + b$, then $K$ is said to be a *radical extension* of $k$. A tower of fields is said to be *radical* if each extension in the tower is radical. Generally a *radical field* $L$ over $k$ is an extension of $k$ for which there exists a radical tower of fields between $L$ and $k$. (Note that we differ from a common usuage of the term "radical extension" which refers to a purely inseparable extension.) An element of a radical extension of $k$ can always be written in terms of (possibly nested) radicals.

The work contained in this paper comes from the author's thesis (ref. 5). The proofs of the theorems quoted in this paper can be found there.

## Gullible and Conservative Radical Simplifiers

The problem into which the gullible system fell, and which the conservative system avoided, can be characterized by the following transformation: $\sqrt{AB} \rightarrow \sqrt{A}\sqrt{B}$. Assuming all square roots take the same branch and all occurrences of a single radical refer to the same element (assumptions which will be maintained throughout this paper) this transformation is valid if and only if $\arg AB = \arg A + \arg B$. The correct transformation is

$$\sqrt{AB} \rightarrow e^{i(\arg AB - \arg A - \arg B)/2} \sqrt{A}\sqrt{B}.$$

Thus we have $\sqrt{(-1)(-1)} = -\sqrt{-1}\sqrt{-1} = (-1)(-1) = 1$ as desired.

In general this transformation takes the following form:

$$\sqrt[r]{A_1 A_2 \dots A_m} \rightarrow e^{i(\arg A_1 \dots A_m - \arg A_1 - \dots - \arg A_m)/r} \sqrt[r]{A_1} \dots \sqrt[r]{A_m}.$$

Similar expressions are valid for logarithms but their consideration would take us a bit far afield. It should be noted that for algebraic functions there are other techniques which may be useful. For instance, we might want to know under what circumstances $\sqrt{AB} = \sqrt{A}\sqrt{B}$, where $A$ and $B$ are functions of $x$. This may be a valid transformation for $x$ in a certain region, in which case restricting $x$ to that region may be the appropriate course of action.

From now on we shall assume that in implementing the techniques outlined below sufficient care is taken with regard to the problems just mentioned. This is not too difficult and we shall point out the one point at which care must be exercised.

## Construction of the Basis

Assume $K$ is a radical extension of $k$ of degree $m$. Then $K$ is an $m$-dimension $k$-vector space. We propose to find a set of elements $\{\alpha_1, \ldots, \alpha_m\}$ contained in $K$, linearly independent over $k$, which spans $K$. Then all the elements of $K$ may be expressed as:

$$\omega_1\alpha_1 + \omega_2\alpha_2 + \ldots + \omega_m\alpha_m,$$

where the $\omega_i$ are elements of $k$. As an example consider $K = k(\sqrt{2}, \sqrt{3}, \sqrt{6})$, $k = \mathbb{Q}$. We shall see that $[K:k] = 4$ but we have eight candidates for the $\alpha_i$:

$$1, \sqrt{2}, \sqrt{3}, \sqrt{6}, \sqrt{2}\sqrt{3}, \sqrt{2}\sqrt{6}, \sqrt{3}\sqrt{6}, \sqrt{2}\sqrt{3}\ \sqrt{6}.$$

Our algorithm will recognize $\sqrt{6} = \sqrt{2}\sqrt{3}$, thus picking as a basis $1$, $\sqrt{2}$, $\sqrt{3}$, $\sqrt{2}\sqrt{3}$. A more illuminating example is provided by $k = \mathbb{Q}(\sqrt{6})$, $K = k(\sqrt{2}, \sqrt{5 + 2\sqrt{6}})$. Recognizing

$$2(5 + 2\sqrt{6}) = (2 + \sqrt{6})^2,$$

we will use $1$, $\sqrt{2}$ as the basis elements and let

$$\sqrt{5 + 2\sqrt{6}} = \frac{2 + \sqrt{6}}{\sqrt{2}} = \sqrt{2} + \frac{\sqrt{2}\sqrt{6}}{2}.$$

This technique is based on the following result:

Theorem: *Let $K = k(\alpha_1^{1/r}, \ldots, \alpha_n^{1/r})$ and let $\Delta = \{x = \alpha_1^{s_1} \ldots \alpha_n^{s_n} \mid 0 \le s_i \le r$ and $x$ is not a perfect $r^{\text{th}}$ power of an element of $k\}$, then the degree of $K$ over $k$ is the number of elements of $\Delta$.*

It is clear that $[K:k]$ is bounded by the cardinality of $\Delta$ since $K$ contains the set of linear combinations of elements of $k$ and $r^{\text{th}}$ roots of elements of $\Delta$. The theorem says that the elements of $\Delta$ are actually linearly independent. This is precisely the set of basis elements for which we were looking.

In the general problem we have radicals $\beta_1^{1/r_1}, \ldots, \beta_n^{1/r_n}$ which we adjoin to $k$. Let $r$ be the least common multiple of the $r_i$ and let $\alpha_i = \beta_i^{r/r_i}$. So, $K = k(\alpha_1^{1/r}, \ldots, \alpha_n^{1/r})$. Clearly $\Delta_1 = \{\alpha_1^{s_1} \ldots \alpha_n^{s_n} \mid 0 \le s_i < r_i\}$ forms a group under multiplication modulo $\alpha_i^{r_i}$. Some of the elements of $\Delta_1$ may actually be perfect $r^{\text{th}}$ powers as elements of $k$. Any such element generates a subgroup of elements which are perfect $r^{\text{th}}$ powers in $k$. Consider the following example: let $n=3$, $r=6$ and assume all the $r_i$ are also 6. $\Delta_1$ has $6 \times 6 \times 6 = 216$ elements. If we can determine that $\alpha_1^2 \alpha_2^3$ is a perfect sixth power then we also have

$$(\alpha_1^2\alpha_2^3)^2 = \alpha_1^4, \qquad (\alpha_1^2\alpha_2^3)^3 = \alpha_2^4,$$
$$(\alpha_1^2\alpha_2^3)^4 = \alpha_1^2.$$

All are perfect sixth powers that is, $\alpha_1$ is a perfect cube and $\alpha_2$ is a perfect square. This reduces the size of $\Delta_1$ by a factor of 6.

There are many techniques available for finding the "quotient group" as it is called. We present one method which is particularly suggestive in our particular case. With notation as before, the $n$ $\alpha_i$ are of order $r_i$. Let $\alpha_1^{m_1} \dots \alpha_n^{m_n}$ be an element of $\Delta_1$ which is a perfect $r^{th}$ power in $k$. Assume $m_1 \neq 0$. Let $w = m_1/(r_1 - m_1) \bmod r_1$, where the ratio is reduced to lowest terms in $\mathbb{Z}$ (the rational integers) and then the division takes place in the finite field. Then

$$(\alpha_2^{m_2} \dots \alpha_n^{m_n})^w = (\alpha_1^{r_1-m_1})^w = \alpha_1^{m_1}$$

and we have reduced $\alpha_1$'s order to $m_1$. We also know that

$$(\alpha_2^{m_2} \dots \alpha_n^{m_n})^{r_1/m_1} = 1,$$

so we may repeat the procedure with this new smaller expression and obtain further reductions.

To illustrate this technique consider our favorite example:

$$\sqrt{5 + 2\sqrt{6}} - \sqrt{2}. \tag{1}$$

We have $\alpha_1 = 5 + 2\sqrt{6}$, $\alpha_2 = 2$, $k = \mathbb{Q}(\sqrt{6})$. We are looking for perfect squares. $\Delta_1$ is of order 4 and there are only 3 elements to check: $\alpha_1 = 5 + 2\sqrt{6}$, $\alpha_2 = 2$, and $\alpha_1\alpha_2 = 10 + 4\sqrt{6}$. $\alpha_2$ is obviously not a perfect square. For $\alpha_1$ we have to work a bit. Assume it was,

$$5 + 2\sqrt{6} = (a + b\sqrt{6})^2 = a^2 + 6b^2 + 2ab\sqrt{6}$$

where $a$ and $b$ must be rational numbers. The resulting pair of equations must possess solutions in rational numbers. This leads to

$$a^4 - 5a^2 + 6 = (a^2 - 2)(a^2 - 3) = 0$$

which plainly has no rational roots. Thus $\alpha_1$ is not a perfect square in $\mathbb{Q}(\sqrt{6})$. (An alternative manner of determining this is to factor $x^2 - \alpha_1$ over $\mathbb{Q}(\sqrt{6})$ if an algebraic factoring algorithm is available.) By analogous reasoning we deduce that $\alpha_1\alpha_2$ is a

# AUTHOR INDEX

Abdali . . . . . . . . . . . . . 253 3/F10
Ament . . . . . . . . . . . . . 87 2/A7
Andersen . . . . . . . . . . . . 161 2/F11
Anderson . . . . . . . . . . . . 395 5/C8
Avgoustis . . . . . . . . . . . 21 1/C6
Bogen . . . . . . . . . . . 11, 75 1/B10
Bulnes . . . . . . . . . . . . . 461 6/A9
Caviness . . . . . . . . . . . . 253 3/F10
Char . . . . . . . . . . . . . . 53 1/E10
Cohen . . . . . . . . . . . . . 275 4/A9
Cuthill . . . . . . . . . . . . 131 2/E3
Doohovskoy . . . . . . . . . . . 473 6/B7
Fateman . . . . . . . . . . 43, 327 1/D14
Fennelly . . . . . . . . . . . . 97 2/B3
Geddes . . . . . . . . . . . . . 405 5/D4
Genesereth . . . . . . . . 291, 309 4/B11
Golden . . . . . . . . . . . 1, 109 1/A14
Gosper . . . . . . . . . . . . . 237 3/E8
Grünbaum . . . . . . . . . . . . 491 6/C11
Gupta . . . . . . . . . . . . . 151 2/F1
Ivie . . . . . . . . . . . . . . 317 4/D9
Karney . . . . . . . . . . . . . 377 5/B4
Kulp . . . . . . . . . . . . . . 385 5/B12
Lafferty . . . . . . . . . . . . 347 4/F11
Lau . . . . . . . . . . . . . . 395 5/C8
Lewis . . . . . . . . . . . . . 277 4/A11
Moenck . . . . . . . . . . . . . 225 3/D10
Moses . . . . . . . . . . . 123, 275 2/D1
Ng . . . . . . . . . . . . . 151, 177 2/F1
Noor . . . . . . . . . . . . . . 161 2/F11
Pavelle . . . . . . . . . . . 75, 97 1/G4
Poole . . . . . . . . . . . . . 145 2/E9
Pridor . . . . . . . . . . . . . 253 3/F10
Rothstein . . . . . . . . . . . 263 3/G6
Spear . . . . . . . . . . . . . 369 5/A10
Steele . . . . . . . . . . . 203, 215 3/C2
Stoutemyer . . . . . . . 315, 425, 447 4/D7
Wahlquist . . . . . . . . . . . 71 1/F14
Wang . . . . . . . . . . . . 55, 435 1/E12
Yun . . . . . . . . . . . . . . 65 1/F8
Zippel . . . . . . . . . . . . . 361 4/G11

# CONTENTS

FOREWORD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iii 1/A1

PREFACE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . v 1/A12

AUTHOR INDEX . . . . . . . . . . . . . . . . . . . . . . . . . . . . vii 1/A1

1. MACSYMA'S SYMBOLIC ORDINARY DIFFERENTIAL EQUATION SOLVER . . . . . . . 1 1/A14
   Jeffrey P. Golden

2. A PROGRAM FOR THE SOLUTION OF INTEGRAL EQUATIONS . . . . . . . . . . . 11 1/B10
   Richard A. Bogen

3. SYMBOLIC LAPLACE TRANSFORMS OF SPECIAL FUNCTIONS . . . . . . . . . . . 21 1/C6
   Yannis Avgoustis

4. AN IMPROVED ALGORITHM FOR THE ISOLATION OF POLYNOMIAL REAL ZEROS . . . 43 1/D1
   Richard J. Fateman

5. FLOATING POINT ISOLATION OF ZEROS OF A REAL POLYNOMIAL VIA MACSYMA . . 53 1/E10
   Bruce W. Char

6. PRESERVING SPARSENESS IN MULTIVARIATE POLYNOMIAL FACTORIZATION . . . . 55 1/E12
   Paul S. Wang

7. ON THE EQUIVALENCE OF POLYNOMIAL GCD AND SQUAREFREE FACTORIZATION
   PROBLEMS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65 1/F8
   David Y. Y. Yun

8. DIFFERENTIAL FORM ANALYSIS USING MACSYMA . . . . . . . . . . . . . . . 71 1/F14
   Hugo D. Wahlquist

9. INDICIAL TENSOR MANIPULATION ON MACSYMA . . . . . . . . . . . . . . . . 75 1/G4
   Richard A. Bogen and Richard Pavelle

10. PURE FIELD THEORIES AND MACSYMA ALGORITHMS . . . . . . . . . . . . . . 87 2/A7
    William S. Ament

11. BLACK HOLES AND RELATIVISTIC GRAVITY THEORIES . . . . . . . . . . . . . 97 2/B3
    A. J. Fennelly and Richard Pavelle

12. THE EVALUATION OF ATOMIC VARIABLES IN MACSYMA . . . . . . . . . . . . . 109 2/C1
    Jeffrey P. Golden

13. THE VARIETY OF VARIABLES IN MATHEMATICAL EXPRESSIONS . . . . . . . . . 123 2/D1
Joel Moses

14. RATIONAL APPROXIMATION TO $e^{-x}$ WITH NEGATIVE REAL POLES . . . . . . . . 131 2/D9
Elizabeth Cuthill

15. TIMING FORMULAS FOR DISSECTION ALGORITHMS ON VECTOR COMPUTERS . . . . 145 2/E9
W. G. Poole, Jr.

16. SYMBOLIC CALCULATIONS IN A FINITE DYNAMIC ELEMENT ANALYSIS . . . . . 151 2/F1
Kajal K. Gupta and Edward W. Ng

17. SYMBOLIC MANIPULATION TECHNIQUES FOR VIBRATION ANALYSIS OF LAMINATED ELLIPTIC PLATES . . . . . . . . . . . . . . . . . . . . . 161 2/F11
C. M. Andersen and Ahmed K. Noor

18. OBSERVATIONS ON APPROXIMATE INTEGRATIONS . . . . . . . . . . . . . . . 177 2/G13
Edward W. Ng

19. LISP: PROGRAM IS DATA – A HISTORICAL PERSPECTIVE ON MACLISP . . . . 181 3/A8
Jon L White

20. LISP: DATA IS PROGRAM – A TUTORIAL IN LISP . . . . . . . . . . . . . 191 3/B4
Jon L White

21. DATA REPRESENTATIONS IN PDP-10 MACLISP . . . . . . . . . . . . . . . . 203 3/C2
Guy Lewis Steele Jr.

22. FAST ARITHMETIC IN MACLISP . . . . . . . . . . . . . . . . . . . . . . 215 3/C14
Guy Lewis Steele Jr.

23. ON COMPUTING CLOSED FORMS FOR SUMMATIONS . . . . . . . . . . . . . . . 225 3/D10
Robert Moenck

24. INDEFINITE HYPERGEOMETRIC SUMS IN MACSYMA . . . . . . . . . . . . . . . 237 3/E8
R. Wm. Gosper, Jr.

25. MODULAR POLYNOMIAL ARITHMETIC IN PARTIAL FRACTION DECOMPOSITION . . . 253 3/F10
S. K. Abdali, B. F. Caviness, and A. Pridor

26. A NEW ALGORITHM FOR THE INTEGRATION OF EXPONENTIAL AND LOGARITHMIC FUNCTIONS . . . . . . . . . . . . . . . . . . . . . . . 263 3/G6
Michael Rothstein

27. SUMMATION OF RATIONAL EXPONENTIAL EXPRESSIONS IN CLOSED FORM . . . . 275 4/A9
Joel Moses and Jacques Cohen

28. USER AIDS FOR MACSYMA . . . 277 4/A11
V. Ellen Lewis

29. THE DIFFICULTIES OF USING MACSYMA AND THE FUNCTION OF USER AIDS . . . 291 4/B11
Michael R. Genesereth

30. AN AUTOMATED CONSULTANT FOR MACSYMA . . . 309 4/D1
Michael R. Genesereth

31. A MACSYMA COMPUTER-ALGEBRA MOVIE DEMONSTRATION . . . 315 4/D7
David R. Stoutemyer

32. SOME MACSYMA PROGRAMS FOR SOLVING DIFFERENCE EQUATIONS . . . 317 4/D9
John Ivie

33. SOME COMMENTS ON SERIES SOLUTIONS . . . 327 4/E5
Richard J. Fateman

34. POWER SERIES SOLUTIONS OF ORDINARY DIFFERENTIAL EQUATIONS IN MACSYMA . . . 347 4/F11
Edward L. Lafferty

35. RADICAL SIMPLIFICATION MADE EASY . . . 361 4/G11
Richard E. B. Zippel

36. A CONSTRUCTIVE APPROACH TO COMMUTATIVE RING THEORY . . . 369 5/A10
David A. Spear

37. REDUCTION OF THE EQUATION FOR LOWER HYBRID WAVES IN A PLASMA TO A NONLINEAR SCHRÖDINGER EQUATION . . . 377 5/B4
Charles F. F. Karney

38. RAY TRAJECTORIES IN A TORUS: AN APPLICATION OF MACSYMA TO A COMPLEX NUMERICAL COMPUTATION . . . 385 5/B12
John L. Kulp

39. APPLICATION OF MACSYMA TO FIRST ORDER PERTURBATION THEORY IN CELESTIAL MECHANICS . . . 395 5/C8
John D. Anderson and Eunice L. Lau

40. SYMBOLIC COMPUTATION OF RECURRENCE EQUATIONS FOR THE CHEBYSHEV SERIES SOLUTION OF LINEAR ODE's . . . 405 5/D4
K. O. Geddes

41. sin(x)**2 + cos(x)**2 = 1 . . . 425 5/E10
David R. Stoutemyer

42. MATRIX COMPUTATIONS IN MACSYMA . . . 435 5/F6
Paul S. Wang

43. SYMBOLIC COMPUTER VECTOR ANALYSIS . . . . . . . . . . . . . . . . . . . . 447 5/G4
David R. Stoutemyer

44. A NATURAL WAY TO DO SPATIAL LINEAR GEOMETRY IN MACSYMA . . . . . . . . 461 6/A9
Juan Bulnes

45. VARIETIES OF OPERATOR MANIPULATION . . . . . . . . . . . . . . . . . . 473 6/B7
Alexander Doohovskoy

46. PROGRESS REPORT ON THE DETERMINANT OF A RANDOM MATRIX . . . . . . . . . 491 6/C11
F. A. Grünbaum

BIBLIOGRAPHY . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 493 6/C13

perfect square (which was pointed out earlier). Now comes the dangerous part. By taking square roots we get

$$\sqrt{2}\sqrt{5+2\sqrt{6}} = 2+\sqrt{6}.$$

Making the appropriate substitution in (1) we finally get $\frac{\sqrt{6}\sqrt{2}}{2}$ (or $\sqrt{3}$) as desired. An inappropriate choice of the root of unity at this step would be the source of incorrect answers.

At SYMSAC '76, Fateman posed the following problem due to Shanks:

$$\sqrt{11+2\sqrt{29}} + \sqrt{16-2\sqrt{29}+2\sqrt{55-10\sqrt{29}}} = \sqrt{22+2\sqrt{5}}+\sqrt{5}.$$

The triply nested radical is not a square as an element of $Q(\sqrt{29}, \sqrt{55+10\sqrt{29}})$, but as an element of $Q(\sqrt{5}, \sqrt{29}, \sqrt{55+10\sqrt{29}})$ it is:

$$16-2\sqrt{29}+2\sqrt{55-10\sqrt{29}} = (\sqrt{5}+\sqrt{11-\sqrt{29}})^2.$$

In the next section we show how to determine the fields in which to search for perfect powers; what we consider here is the resulting simplification problem:

$$\sqrt{11+2\sqrt{29}} + \sqrt{11-2\sqrt{29}} = \sqrt{22+2\sqrt{5}}. \qquad (2)$$

Using the technique just described, we have $\alpha_1 = 11+2\sqrt{29}$, $\alpha_2 = 11-2\sqrt{29}$, and $\alpha_3 = 22+2\sqrt{5}$. $\alpha_1\alpha_2 = 5$, which happens to be a perfect square in $k$. This gives the following reduction:

$$\sqrt{11-2\sqrt{29}} = \frac{11-2\sqrt{29}}{\sqrt{5}}\sqrt{11+2\sqrt{29}}.$$

Continuing, we get

$$\alpha_1\alpha_3 = 242+4\sqrt{29}+22\sqrt{5}+4\sqrt{5}\sqrt{29} = (\sqrt{5}+11+2\sqrt{29})^2.$$

So finally

$$\sqrt{22+2\sqrt{5}} = 1+\frac{11-2\sqrt{29}}{\sqrt{5}}\sqrt{11+2\sqrt{29}}.$$

And thus all the radicals involved in (2) can be expressed in terms of a single quadratic extension of $Q(\sqrt{5}, \sqrt{29})$.

## Denesting Nested Radicals

The fundemental concept in this section is that of nesting, and in particular, what the nesting level of a field is. Rather than give the rigorous definition of nesting order (which would probably only serve to confuse the reader) we shall rely upon his intuition and the following examples. The fields $k(\sqrt{2})$, $k(\sqrt{2},\sqrt{3})$, $k(\sqrt{5+2\sqrt{6}})$, and $k(\sqrt{1+\sqrt{2}})$

are singly nested over $k$, $k$, $k$, and $k(\sqrt{2})$ respectively. The next to last field is singly nested because it is contained in a field which is singly nested (i.e. the second field). Thus the nesting of a field is roughly the minimal amount of nesting needed to express the most deeply nested expression in the field over a particular ground field. We are not able to compute the minimal nesting level of any field but we are able to prove the following theorem.

Theorem: *Let $E$ be an algebraic extension of $k$ of nesting level $n$ and let $L = E(\alpha^{1/r})$. If $L$ can be expressed with nesting level $n$ then there is an element $\beta$ of a proper subfield of $E$ such that $\alpha\beta$ is a perfect $r^{th}$ power in $K$.*

As an example consider $\sqrt{5 + 2\sqrt{6}}$. Then $k = \mathbb{Q}$, $E = \mathbb{Q}(\sqrt{6})$. The only proper subfield of $E$ is $\mathbb{Q}$. Thus we have $\beta = 2$ or $3$ since $2(5+2\sqrt{6}) = (2+\sqrt{6})^2$ and $3(5+2\sqrt{6}) = (3+\sqrt{6})^2$. In the general quadratic case we have

$$\beta(p + \sqrt{q}) = (a_0 + a_1\sqrt{q})^2.$$

Since $a_0$ and $\beta$ are elements of a field we may assume $a_1 = 1$ and we have the equations

$$\beta p = a_0^2 + q, \qquad \beta = 2a_0$$

or

$$\beta^2 - 4\beta p + 4q = 0.$$

Since $\beta$ must be rational $p^2 - q$ must be a perfect square. Letting $d^2 = p^2 - q$, we have the following classical formula:

$$\sqrt{p + \sqrt{q}} = \sqrt{\frac{p+d}{2}} + \sqrt{\frac{p-d}{2}}.$$

It is easy to extend this technique to arbitrary degree extensions of $k$. From a practical point of view, however, the systems of equations can become quite unwieldy when the degree is much above 3. The author's thesis contains a number of general formulas which were derived in this manner, but with quite a bit of work. For instance:

$$\sqrt[3]{(m^2 + mn + n^2)\sqrt[3]{(m-n)(m+2n)(2m+n)} + 3mn^2 + n^3 - m^3}$$
$$= \sqrt[3]{\frac{(m-n)(m+2n)^2}{9}} - \sqrt[3]{\frac{(2m+n)(m-n)^2}{9}} + \sqrt[3]{\frac{(m+2n)(2m+n)^2}{9}}.$$

## Conclusions

We have hoped to point out that what had been thought to have been difficult problem, the simplification of nested radicals, is actually not very much more difficult than simplification of un-nested radicals. Of the algorithms presented only the de-nesting algorithm is really very costly, and that algorithm is really not necessary. All the results mentioned here are either classical or direct corollaries of classical results. What we hope to have contributed is a novel way of looking at classical mathematics.

## REFERENCES

1. Kleiman, S.L.: "Computing with Rational Expressions in Several Algebraically Dependent Variables." Bell Laboratories Tech. Rep., 1966.

2. Caviness, B.F.: *On Canonical Forms and Simplification*, Ph.D. Diss., Carnegie-Mellon Univ. 1968.

3. Fateman, R.J.: *Essays in Algebraic Simplification*. Ph.D. Diss., Harvard Univ. 1971. (Revised version available as MIT Tech. Rep. MAC TR-95 1972.

4. Caviness, B.F. and Fateman, R.J.: Simplification of Radical Expressions. Proceedings of the 1976 Symp. on Symbolic and Algebraic Computation, Aug. 1976, pp. 329-338.

5. Zippel, R.E.B.: "Simplification of Nested Radicals with Applications to Solving Polynomial Equations." M.S. Thesis, Mass. Inst. of Tech., 1977.

BLANK PAGE

BLANK PAGE

# A CONSTRUCTIVE APPROACH TO COMMUTATIVE RING THEORY

David A. Spear

Massachusetts Institute of Technology

## 1. INTRODUCTION

We are building , in MACSYMA , a system for Commutative Ring Theory . The object is to determine how much of the theory of commutative rings can be made effective , and to realize those parts of the theory on a computer . We adopt 2 basic goals :

(1) to provide a language capability .

(2) to provide a problem-solving capability .

Our main interest is in solving ring theory problems ; however it is clearly desirable to be able to express information in a language reasonably close to that of ring theory . We present here an outline of the system as we envision it . The implementation has just begun and is proceeding rapidly but as of now only a small part of the system is ready for use .

## 2. ADMISSIBLE RINGS

By an admissible ring , we mean a ring which is allowable in our system . As the system grows , the class of admissible rings will expand . Some axioms of admissibility are :

(1) Z is admissible (Z denotes the integers) .

(2) If R is admissible so is R[X] .

(3) If R is admissible and I is a finitely generated ideal of R then R/I is admissible .

(4) If R is admissible and R is an integral domain then the quotient field of R is admissible .

(5) If R and S are admissible so is their direct sum .

(6) If R and S are admissible so is their tensor product (over Z) .

(7) If R is admissible so is any finitely generated subring of R .

The smallest class of rings satisfying these axioms we shall call the elementary rings . Thus we have

$\{$elementary rings$\} \subset \{$admissible rings$\}$

Initially , all admissible rings will be elementary .

Examples of elementary rings :

(1) $Q [X] // [X^2 - 2]$ (the field $Q( 2^{1/2} )$ )

(2) $Z [X] // [X^2 + 1]$ (the ring of Gaussian integers)

(3) $Z_5 [A,B,C] // [A^2 B - C^7 , A^2 B^3 C^4]$

(4) $Q [X,Y] // [X^2 - Y^3]$

(5) $Q (X) [Y] // [Y^2 - X]$

It should be apparent that the elementary rings form a large and interesting class of rings .

## 3. ALGORITHMS FOR ELEMENTARY RINGS

Built into the system are a collection of algebraic algorithms which work in any elementary ring . Some of these algorithms are classical , others fairly recent , and some , due to the author , are apparently new . In developing the system , most of our energy has been directed toward enlarging and improving its package of algorithms . To give an idea of the strength of the system , we list some of the problems which it is able to solve .

Let R be an elementary ring and let $a_1 , \ldots , a_n \in R$ .

Let I be the ideal of R generated by the $a_i$ and let

S be the subring of R generated by the $a_i$ .

(1) ideal membership .

Given $r \in R$ decide whether or not $r \in I$ .

(2) subring membership .

Given $r \in R$ decide whether or not $r \in S$ .

(3) syzygies .

Find all solutions $x_1 , \ldots , x_n \in R$ to the equation

$$a_1 x_1 + \ldots + a_n x_n = 0$$

(4) algebraic relations .

Find all algebraic relations between $a_1 , \ldots , a_n$ .

(5) prime test .

Decide if I is a maximal ideal .

Decide if I is a prime ideal .

Decide if I is a radical ideal .

(6) dimension .

Compute the dimension of R . (Krull dimension) .

Compute the transcendence degree of R over S .

(7) ideal intersection .

Given ideals I and J compute their intersection .

(8) ideal contraction .

Compute the intersection of the ideal I with the subring S .

(9) units .

Given $r \in R$ decide if r is a unit in R . If so , compute $1/r$ .

(10) zero-divisors .

Given $r \in R$ decide if r is a zero-divisor in R .

If so , find $s \in R$ , $s \neq 0$ , such that $r\,s = 0$ .

## 4. THE CANONICAL FORM

The solution to each of the problems described above depends on a fundamental algorithm for expressing ideals in a canonical form . This algorithm appears to have been first discovered by Buchberger (ref. 1) . Similar algorithms have been constructed by Richman (ref. 2) , Shtokhamer (ref. 3) , and Lauer (ref. 4) . My own version , independently obtained , is only slightly different from Buchberger's ; however the difference is crucial - it is the key to solving most of the problems listed in the previous section . The canonical form for an ideal I is denoted IDEALBASIS (I) . IDEALBASIS has been implemented by David R. Barton .

## 5. EXAMPLES

We give some concrete examples , illustrating the use of the system :

```
(C1) R: RING( Q [X,Y] // [X^2 - Y^3] ) ;

                                             2    3
(D1)                         Q [X,Y] // [X   - Y ]

(C2) DOMAINP (R) ;

(D2)                                TRUE

(C3) FIELDP (R) ;

(D3)                                FALSE

(C4) DIMENSION (R) ;

(D4)                                 1

(C5) I: IDEAL ( [X] , R ) ;

(D5)                                [X]

(C6) RADICALP (I) ;

                                   3
(E6)                               Y  ● I

(E7)                             Y  NOT ●  I

(D7)                                FALSE

(C8) RADICAL (I) ;

(D8)                               [X,Y]
```

```
(C9) R· RING ( Z [X] // [X^2  - 2] ) ;


                                   2
(D9)                    Z [X] // [X  - 2]

(C10)  I: IDEAL ( [7] , R ) ;

(D10)                         [7]

(C11) PRIMEP (I,R) ;

(E11)                 (3 + X)  (3 - X) ●  I

(E12)                    (3 + X)  NOT ●  I

(E13)                    (3 - X)  NOT ●  I

(D13)                          FALSE

(C14) UNIT( 3 + 2 * X , R ) ;

(E14)                (3 + 2 X)  (3 - 2 X) = 1

(D14)                          TRUE

(C15) R: RING ( Q [X,Y] ) ;


(D15)                        Q [X,Y]

(C16) I: IDEAL ([X^3 * Y^4 , X^2 * Y^6] , R) ;


                         3   4      2  6
(D16)                   [X  Y    , X  Y ]

(C17) J: IDEAL ([X * Y^9 , X^5 * Y] , R) ;

                             9    5
(D17)                     [X Y  , X  Y]

(C18) INTERSECTION (I,J) ;

                          2  9    5  4
(D18)                    [X  Y  , X  Y ]
```

```
(C19) S: SUBRING ( Q [X + Y, X * Y] , R ) ;

(D19)                       Q [X + Y , X Y]

(C20) MEMBER ( X^2 + Y^3 , S ) ;

(D20)                             FALSE
```

## 6. FUTURE PLANS

Within the next year , many improvements and additions to the system are likely . For example , we plan to allow R-modules into the system . Algebraic Number Theory and Algebraic Geometry offer other possible directions for the system . However , much of the growth of the system will be determined by the needs of its users . We welcome suggestions for changes or new features .

A complete , current desciption of the Ring Theory System can be found on the MC file :

DAS; RINGS INFO

This file also contains descriptions of system commands , examples , and other information relevant to the use of the system .

I would like to thank David R. Barton for his excellent implementation of IDEALBASIS . I would also like to thank Alex P. Doohovskoy and Barry M. Trager for their encouragement and for many helpful suggestions .

# REFERENCES

1. Buchberger , B . : Ein Algorithmisches Kriterium Fur Algebraischen Gleichungssystems . Aequationes Mathematicae , vol. 4 , 1970 .

2. Richman , F . : Constructive Aspects of Noetherian Rings . Proc. American Math. Soc. , vol. 44 , 1974 , pp. 436 - 441 .

3. Shtokhamer , R . : Simple Ideal Theory : Some Applications to Algebraic Simplification . Tech. Rep. UCP-36 , Univ. of Utah , 1975 .

4. Lauer , M . : Canonical Representatives for Residue Classes of a Polynomial Ideal . Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation , Aug . 1976 , pp. 339 - 345 .

# Reduction of the Equation for Lower Hybrid Waves in a Plasma to a Nonlinear Schrödinger Equation[*]

by

Charles F. F. Karney

Research Laboratory of Electronics and Plasma Fusion Center,
Massachusetts Institute of Technology

The equations describing the nonlinear propagation of waves in an anisotropic plasma are rarely exactly soluble. However it is often possible to make approximations that reduce the exact equations into a simpler equation. In this paper we will describe how MACSYMA may be used to make such approximations, and so reduce the equation describing lower hybrid waves into the nonlinear Schrödinger equation which is soluble by the inverse scattering method (ref. 1). It should be pointed out here that we have not used MACSYMA to do the whole problem; rather MACSYMA is used at several stages in the calculation that are otherwise done by hand. This is not to say that MACSYMA could not do the whole problem, just that there is a natural division between calculations that are easiest done by hand, and those that are easiest done by machine.

The equation describing the steady-state two-dimensional electrostatic propagation of lower hybrid waves in a homogeneous magnetized plasma is (refs. 2, 3)

$$\begin{aligned} K_\perp \frac{\partial^2}{\partial x^2}\phi - |K_\parallel| \frac{\partial^2}{\partial z^2}\phi + a\frac{\partial^4}{\partial x^4}\phi + b\frac{\partial^4}{\partial x^2 \partial z^2}\phi + c\frac{\partial^4}{\partial z^4}\phi \\ + \frac{\epsilon_0}{4}\alpha_0 \frac{\partial}{\partial x}\left[\frac{|\nabla\phi|^2}{n_0 T}\frac{\partial}{\partial x}\phi\right] + \frac{\epsilon_0}{4}\beta_0 \frac{\partial}{\partial z}\left[\frac{|\nabla\phi|^2}{n_0 T}\frac{\partial}{\partial z}\phi\right] = 0\,, \end{aligned} \tag{1}$$

where $\phi$ is the complex potential and $x$ and $z$ are the directions parallel and perpendicular to the magnetic field and the other quantities are constants. (The real potential is $\mathrm{Re}[\phi \exp(-i\omega t)]$, where $\omega$ is the frequency of the wave.) The significance of the terms in equation (1) is as follows: The first two terms (with coefficents, $K_\perp$ and $|K_\parallel|$) describe the linear, cold, electrostatic response; they constitute a wave equation and have solutions which propagate along well defined rays (ref. 4). The terms with coefficients $a$, $b$, and $c$ in equation (1) are the corrections due to the finite temperature of the plasma; the effect of these terms is to cause the ray to disperse. The terms on the second line (with coefficients $\alpha_0$ and $\beta_0$) are due to the nonlinearity of the plasma; these terms arise because in regions where the electric potential is high, the so-called ponderomotive force expels some of the plasma, causing a change in the dielectric properties of the medium.

We wish to reduce equation (1) to a more manageable form. To do this we must decide what type of solution we are looking for. Since we are interested in situations where the nonlinear terms are perturbations to the linear terms, and since wave-like solutions are known for linear problems, interesting solutions to consider are ones of the form

---

[*] Work supported by U.S. Energy Research and Development Administration (Contract E(11-1)-3070) and by the National Science Foundation (Contract ENG76 06242)

$$\phi(x, z) = \Phi(x, z)\exp(ik_z z - ik_x x) , \tag{2}$$

where the wavenumbers $k_x$ and $k_z$ are constants and the complex envelope, $\Phi$, is slowly varying compared with the exponential. Since we wish to treat the nonlinear terms as a perturbation, we need only consider the leading order contributions to these terms. Thus we can immediately simplify the nonlinear terms since each derivative operator will bring down either $ik_z$ or $-ik_x$; thus they may be written as

$$\frac{\epsilon_0}{4}\alpha_0\frac{\partial}{\partial x}\left[\frac{|\nabla\phi|^2}{n_0T}\frac{\partial}{\partial x}\phi\right] + \frac{\epsilon_0}{4}\beta_0\frac{\partial}{\partial z}\left[\frac{|\nabla\phi|^2}{n_0T}\frac{\partial}{\partial z}\phi\right] = C|\Phi|^2\Phi\exp(ik_z z - ik_x x) , \tag{3}$$

where $C$ is a constant. The problem remaining is to reduce the complexity of the linear terms. This we can do by saying that the dispersion has only a weak effect on the solution (in the final equation we will see that the nonlinearity and dispersion are treated as being perturbations of the same order). If we neglect dispersion entirely, then a solution for $\Phi$ is

$$\Phi(x, z) = \Phi(z - v_g x) ; \tag{4}$$

i.e. the waves travel along characteristics. We will treat the effects of both dispersion and nonlinearity by letting $\Phi$ have an explicit $x$ dependence; thus

$$\phi(x, z) = \Phi(z', x')\exp(ik_z z - ik_x x) , \tag{5}$$

where $z' = z - v_g x$, $x' = x$. We order the dependencies in equation (5) as follows

$$|ik_x| \gg |v_g\partial/\partial z'| \gg |\partial/\partial x'| , \quad |ik_z| \gg |\partial/\partial z'| . \tag{6}$$

[This ordering is not the only possible one; for instance Morales and Lee (ref. 2) considered the case where $k_x = k_z = 0$, and derived a modified Korteweg–deVries equation.]

Rather than using this ordering directly in equation (1), it is more convenient to treat the more general problem. So we re-write the linear terms in equation (1), to give

$$L(\frac{\partial}{\partial x}, \frac{\partial}{\partial z})\phi + \text{nonlinear terms} = 0 , \tag{7}$$

where $L$ is a polynomial,

$$L(p, q) = K_\perp p^2 - |K_\parallel|q^2 + ap^4 + bp^2q^2 + cq^4 . \tag{8}$$

Now if $L(\partial/\partial x, \partial/\partial z)$ operates on equation (5) we may make the replacements

$$\partial/\partial x \to -ik_x - v_g\partial/\partial z' + \partial/\partial x' , \quad \partial/\partial z \to ik_z + \partial/\partial z' . \tag{9}$$

We may then Taylor expand $L$ about $-ik_x$ and $ik_z$. This is, of course, most easily done on MACSYMA:

```
(C1) GRADEF(L(P,Q),L1(P,Q),L2(P,Q))$

(C2) GRADEF(L1(P,Q),L11(P,Q),L12(P,Q))$

(C3) GRADEF(L2(P,Q),L12(P,Q),L22(P,Q))$
```

Unfortunately MACSYMA has no notation for the derivative of a function with respect to its arguments; thus we use GRADEF to define L1 to denote the derivative of L with respect to its first argument, etc.

```
(C4) L(P,Q);
(D4)                                   L(P, Q)

(C5) %,P=-%I*KX-ZEPS*VG*DZ1+ZEPS^2*DX1,Q=%I*KZ+ZEPS*DZ1;
                          2
(D5)            L(DX1 ZEPS  - DZ1 VG ZEPS - %I KX, DZ1 ZEPS + %I KZ)
```

Here we have just written L(P,Q), substituted for P (= $\partial/\partial x$) and Q (= $\partial/\partial z$) using equation (9). In order to incorporate the ordering information implied by equation (6) we have introduced the small parameter ZEPS. (ZEPS is chosen rather than, say, EPS, since MACSYMA will treat it as the main variable in CRE forms.) DZ1 and DX1 are used to denote $\partial/\partial z'$ and $\partial/\partial x'$ respectively.

```
(C6) TAYLOR(%,ZEPS,0,2)$

(C7) LEXPAND:EV(%,L(-%I*KX,%I*KZ)=L,
                L1(-%I*KX,%I*KZ)=L1,
                L2(-%I*KX,%I*KZ)=L2,
                L11(-%I*KX,%I*KZ)=L11,
                L12(-%I*KX,%I*KZ)=L12,
                L22(-%I*KX,%I*KZ)=L22);
                 2       2        2             2                     2
(D7)/R/ 1/2 ((DZ1  L11 VG  - 2 DZ1  L12 VG + DZ1  L22 + 2 DX1 L1) ZEPS

                                     + (- 2 DZ1 L1 VG + 2 DZ1 L2) ZEPS + 2 L)
```

We carry out the Taylor expansion using TAYLOR, keeping terms up to ZEPS^2. The result, LEXPAND, is made more compact by making the functional dependence of L on KX and KZ implicit.

Since we are interested in the balance of the nonlinear term, equation (3), against the dispersive part of the linear operator, *L*, we demand that all but the ZEPS^2 term in D7 vanish identically. (Note that the the ZEPS^2 term contains the dispersive operator, $\partial^2/\partial z'^2$.)

```
(C8) LEXPANDO:COEFF(LEXPAND,ZEPS,0);
(D8)/R/                                L
```

The zeroth order term is just $L(-ik_x, ik_z)$. Setting it to zero

$$L(-ik_x, ik_z) = 0 \tag{10}$$

just states that $k_x$ and $k_z$ must satisfy the linear dispersion relation.

```
(C9) LEXPAND1:COEFF(LEXPAND,ZEPS,1);
(D9)/R/                        - DZ1 L1 VG + DZ1 L2

(C10) SOLVE(LEXPAND1=0,VG);
```

```
SOLUTION
                                                        L2
(E10)                                             VG = --
                                                        L1
(D10)                                              [E10]
```

Setting the first order term to zero gives us the expression for $v_g$. We recognize E10 as the familiar expression for the group velocity in a dispersive medium,

$$v_g = \frac{L_q}{L_p}\bigg|_{p = -ik_x,\ q = ik_z} . \tag{11}$$

(The subscripts $p$ and $q$ denote derivatives.)

```
(C11) LEXPAND2:COEFF(LEXPAND,ZEPS,2);
                     2          2           2                2
(D11)/R/    1/2 (DZ1  L11 VG  - 2 DZ1  L12 VG + DZ1  L22 + 2 DX1 L1)

(C12) AA:COEFF(LEXPAND2,DX1);
(D12)/R/                                  L1

(C13) BB:COEFF(LEXPAND2,DZ1,2);
                                     2
(D13)/R/                   1/2 (L11 VG  - 2 L12 VG + L22)
```

Finally we have the order ZEPS^2 terms. Note that it has the form $A\partial/\partial x' + B\partial^2/\partial z'^2$, where $A$ and $B$ are given by (AA in D12 and BB in D13)

$$A = L_p,\ B = \tfrac{1}{2}L_{pp}v_g^2 - L_{pq}v_g + \tfrac{1}{2}L_{qq} . \tag{12}$$

(All the derivates are evaluated at $p = -ik_x$, $q = ik_z$.) If we demand that the ZEPS^2 term balance the nonlinear term, equation (3), we obtain

$$A\Phi_{x'} + B\Phi_{z'z'} + C|\Phi|^2\Phi = 0 . \tag{13}$$

If $A$ is pure imaginary and $B$ and $C$ are real (which turns out to be the case) then equation (13) is the nonlinear Schrödinger equation.

The last task is to evaluate the coefficients $A$ and $B$, for $L$ given by equation (8). Again, in order to get manageable expressions, we will do this approximately. This time we note that the coefficients, $a$, $b$, and $c$ are much smaller than $K_\perp$ and $K_\parallel$. Again such manipulations are most readlily performed on MACSYMA:

```
(C14) L:KPERP*P^2+KPAR*Q^2+ZDTA*(A*P^4+B*P^2*Q^2+C*Q^4);
                     4       2  2        4                  2          2
(D14)            (C Q  + B P  Q  + A P ) ZDTA + KPAR Q  + KPERP P

(C15) (L1:DIFF(L,P),
       L2:DIFF(L,Q),
       L11:DIFF(L1,P),
       L12:DIFF(L1,Q),
       L22:DIFF(L2,Q),
VG:EV(RHS(E10)))$
```

Here we have defined L [see eq. (8)]. The smallness of *a*, *b*, and *c* is implied by the small parameter ZDTA. We have also defined the various derivatives of L, and VG. The evaluation of *A* (AA) is straightforward. We Taylor expand AA to obtain the leading term.

```
(C16) AA:EV(AA,P=-%I*KX,Q=%I*KZ,EVAL);
                           2            3
(D16)/R/      (2 %I B KX KZ  + 4 %I A KX ) ZDTA - 2 %I KPERP KX

(C17) AA:TAYLOR(AA,ZDTA,0,0);
(D17)/T/                 - 2 KPERP %I KX + . . .
```

I.e.

$$A = -2ik_x K_\perp . \tag{14}$$

We repeat this with *B* (BB).

```
(C18) BB:EV(BB);
                2  8          2      2     2  6                  3   4  4
(D18)/R/ ((4 B C  Q  + (24 A C  + 2 B  C) P  Q  + (32 A B C - 2 B ) P  Q

        2          2   6  2      2    8      3
 + (24 A  C + 2 A B ) P  Q  + 4 A  B P ) ZDTA

        2                      6                             2         2  4
 + ((4 C  KPERP + 4 B C KPAR) Q  + (8 B C KPERP + (24 A C - B ) KPAR) P  Q

               2                       4  2                     2        6
 + ((24 A C - B ) KPERP + 8 A B KPAR) P  Q  + (4 A B KPERP + 4 A  KPAR) P )

     2                            2   4             2           2   2  2
 ZDTA  + ((4 C KPAR KPERP + B KPAR ) Q  + (6 C KPERP  + 6 A KPAR ) P  Q

           2                    4             2        2             2  2
 + (B KPERP  + 4 A KPAR KPERP) P ) ZDTA + KPAR  KPERP Q  + KPAR KPERP  P )

    2  2  4          4  2      2  6      2
/((B  P  Q  + 4 A B P  Q  + 4 A  P ) ZDTA

               2  2              4              2  2
 + (2 B KPERP P  Q  + 4 A KPERP P ) ZDTA + KPERP  P )

(C19) BB:TAYLOR(BB,ZDTA,0,1);
             2  2               2
         KPAR  Q  + KPAR KPERP P           2                      4
(D19)/T/ ------------------------- - ((KPAR  B - 4 KPAR C KPERP) Q
                       2
                KPERP P
```

```
                           2              2    2  2        2    4
 + (2 KPAR KPERP B - 2 KPAR  A - 6 C KPERP ) P  Q  - KPERP  B P ) ZDTA

       2  2
/(KPERP  P ) + . . .
```

Note that we have taken the Taylor series expansion of BB up to order ZDTA. This is because the order ZDTA^0 term is proportional to the order ZDTA^0 terms in L (see D14), and thus when we set *L* to zero [see eq. (10)] the leading order term will vanish (this is just a reflection of the fact that "cold" contributions to *L*, $K_\perp p^2 - |K_\parallel| q^2$, are non-dispersive). There are a number of ways of incorporating the fact that L = 0 into D19; we chose the following:

```
(C20) SOLVE(L=0,KPERP);
SOLUTION
                                   4      2  2      4               2
                             (C Q  + B P  Q  + A P ) ZDTA + KPAR Q
(E20)           KPERP = - ----------------------------------------
                                                2
                                               P
(D20)                                  [E20]

(C21) BB:EV(BB,E20)$

(C22) BB:TAYLOR(BB,ZDTA,0,1);
                           4        2  2        4
                   (3 C Q  + 3 B P  Q  + 3 A P ) ZDTA
(D22)/T/           ---------------------------------- + . . .
                                     2
                                    Q
```

(Note that indeed the coefficient of ZDTA^0 is zero.)

```
(C23) BB:EV(BB,ZDTA=1,P=-%I*KX,Q=%I*KZ,FACTOR);
                             4        2   2        4
                      3 (C KZ  + B KX   KZ  + A KX )
(D23)               - ------------------------------
                                      2
                                    KZ
```

(Here we have just substituted for *p* and *q*.) Thus

$$B = -\frac{3}{k_z^2}(ak_x^4 + bk_x^2k_z^2 + ck_z^4)\,. \tag{15}$$

Finally a scale transformation on $\Phi$, $x'$, and $z'$ in equation (13) yields

$$iv_\tau + v_{\xi\xi} + 2|v|^2v = 0\,, \tag{16}$$

a standard form of the nonlinear Schrödinger equation.

We could have saved some steps in the MACSYMA computation had we worked with the explicit form of L (D14) right from the beginning. However this would have had the disadvantage of confusing the two small parameters in the problem (ZEPS and ZDTA). Also some of the generality of the method would be lost. For instance, a simple extension of the method outlined above to include the effects of a third spatial dimension [which introduces a term, $K_\perp \partial^2\phi/\partial y^2$ in eq. (1)] is possible (ref. 5). This leads to an unusual generalization of the nonlinear Schrödinger equation,

$$iv_\tau + v_{\xi\xi} - v_{\eta\eta} + 2|v|^2 v = 0 . \tag{17}$$

The procedure presented here was suggested by the work of Newell and Kaup (ref. 6), who use a more traditional multiple-time-scales approach. The help of F. Y. F. Chu in preparing this paper is gratefully acknowledged.

## REFERENCES

1. Scott, A. C.; Chu, F. Y. F.; and McLaughlin, D. W.: *The Soliton: A New Concept in Applied Science*, Proc. IEEE, vol. 61, no. 10, Oct. 1973, pp. 1443-1483.
2. Morales, G. J.; and Lee Y. C.: *The Nonlinear Filamentation of Lower-Hybrid Cones*, Phys. Rev. Lett., vol. 35, no. 14, Oct. 1975, pp. 930-933.
3. Karney, C. F. F.; Chu, F. Y. F.; Johnston, G. L.; and Bers, A.: *Solution to Boundary Value Problem for Propagation of Lower Hybrid Waves*, Plasma Res. Rep. 76/11, Research Laboratory of Electronics, Massachusetts Institute of Technology, Jan. 1976
4. Briggs, R. J.; and Parker, R. R.: *Transport of RF Energy to the Lower Hybrid Resonance in an Inhomogeneous Plasma*, Phys. Rev. Lett., vol. 29, no. 13, Sep. 1972, pp. 852-855.
5. Sen, A.; Karney, C. F. F.; Johnston, G. L.; and Bers, A.: *Three-Dimensional Effects in the Nonlinear Evolution of Lower Hybrid Cones*, Paper D17, Proc. Annual Controlled Fusion Theory Conference, San Diego, May 1977.
6. Newell, A. C.; and Kaup, D. J.: *Nonlinear Propagation Along Lower Hybrid Cones*, Paper 4F-15, Bull. American Phys. Soc., vol. 21, no. 9, Oct. 1976, p. 1095.

BLANK PAGE

BLANK PAGE

# Ray Trajectories in a Torus: An Application of MACSYMA to a Complex Numerical Computation*

by

John L. Kulp

Research Laboratory of Electronics and Plasma Fusion Center,
Massachusetts Institute of Technology

The study of ray trajectories of plasma waves in a toroidal geometry using MACSYMA is an example of how symbolic, numerical, and graphical facilities can be used in concert to accomplish a complex computational goal. Computational features of this study which are of particular significance include: the derivation of code (i.e. writing functions to generate program fragments), the use of array functions to simplify the specification of a numerical iteration scheme, and the graphical presentation of the results. Mathematically, this study originates in the solution of a linear inhomogeneous partial differential equation in 3 dimensions by the method of characteristics. It is possible to describe this equation compactly by using vector notation, and by specifying the spatial variation of the coefficients in terms of intermediate parameters. However the transformation of the equation into a form amenable to solution is very tedious.

This work is part of a study of the heating of plasmas by radio frequency waves occurring in controlled thermonuclear fusion research (ref. 1). The objective is to obtain a description of the rf field structure excited by a waveguide located at the edge of a toroidal plasma confinement device. A steady-state, single frequency driven oscillation is assumed and an examination is made of the resulting spatial distribution of fields. In the electrostatic approximation, the electric potential is then given by

$$\nabla \cdot \mathbf{K}(\mathbf{r}) \cdot \nabla \phi(\mathbf{r}) = D(\nabla,\mathbf{r})\,\phi(\mathbf{r}) = 0$$

where $\mathbf{r}$ is a spatial position vector and $\mathbf{K}$ is a second rank dielectric tensor. For the parameter range of interest, this second order equation is hyperbolic, and its characteristic surfaces $\psi(\mathbf{r})=\text{const}$ can be found from the characteristic form, $D(\nabla\psi,\mathbf{r})=0$ (ref. 2). This nonlinear first order equation can be solved by integrating $\nabla\psi$ along the characteristics of $D(\nabla\psi,\mathbf{r})$ which are rays in 3-dimensions. Unfortunately, transforming to the coordinates given by $\nabla\psi$ does not, in general, reduce the order of $D(\nabla,\mathbf{r})$ since it is a second order operator in 3 independent variables. Thus some additional assumptions are necessary to make the calculation of $\phi$ tractable. If there is a spatial coordinate along which D is uniform, a Fourier decomposition of $\phi$ with respect to that coordinate is usually successful in reducing the number of dimensions of the equation. However, this may be inconvenient for other reasons, such as difficulty in applying initial conditions, or in integrating the resulting Fourier spectrum. An alternative is to pursue solutions in the WKB approximation which have the form,

*Work supported by U.S. Energy Research and Development Administration
(Contract E(11-1)-3070)

$$\phi(\mathbf{r}) = \tilde{\phi}(\mathbf{r})\, e^{i\psi(\mathbf{r})}$$

and where $|\nabla \log\tilde{\phi}| \ll |\nabla\psi|$ is assumed. The former approach has been investigated (ref. 3) for a straight cylinder geometry. Here, the WKB approach is followed since it is more readily generalized (computationally) to models resulting in higher order equations.

In the following sections, (1) a description of the method for finding $\tilde{\phi}$ and $\psi$ is given, (2) the implementation of the calculation on MACSYMA is presented, and (3) a sample case is shown to illustrate the display of results.

## WKB Solution Along the Characteristic Rays.

Let $\mathbf{k} = \nabla\psi$. The characteristic equation $D(\mathbf{k},\mathbf{r})=0$ by itself is not sufficient to determine $\mathbf{k}$. More information can be obtained by noting that

$$\frac{dD(\mathbf{k}(\mathbf{r}),\mathbf{r})}{d\mathbf{r}} = 0 \text{ is also implied so that } \frac{\partial D}{\partial \mathbf{k}} \cdot \frac{\partial \mathbf{k}}{\partial \mathbf{r}} = -\frac{\partial D}{\partial \mathbf{r}} .$$

Thus by integrating along $\partial D/\partial\mathbf{k}$ we can find $\mathbf{k}$. The initial values of 2 components of $\mathbf{k}$ are required (the third can be found from $D(\mathbf{k},0)=0$). The rays defined by the tangent vector $\partial D/\partial\mathbf{k}$ are the bi-characteristics of D. Let s be the distance along the ray from some starting point and $S = |\partial D/\partial\mathbf{k}|$. Then, the equations for determining $\psi$ become:

$$\frac{d\mathbf{r}}{ds} = \frac{\partial D}{\partial \mathbf{k}} \,/\, S \qquad \text{(trajectory equation)}$$

$$\frac{d\mathbf{k}}{ds} = -\frac{\partial D}{\partial \mathbf{r}} \,/\, S \qquad \text{(Wave vector equation)}$$

$$\frac{d\psi}{ds} = (\mathbf{k} \cdot \frac{\partial D}{\partial \mathbf{k}}) \,/\, S \qquad \text{(phase equation)}$$

For the electrostatic equation, note that $\mathbf{k}\cdot\partial D/\partial\mathbf{k} = 0$, so the rays are lines of constant $\psi$. In wave propagation terminology, $\partial D/\partial\mathbf{k}$ is in the direction of the group velocity of the excited waves.

To solve for $\phi$, let $\phi(\mathbf{r}) = \tilde{\phi}(\mathbf{r})e^{i\psi(\mathbf{r})}$ so that

$$D(\nabla,\mathbf{r})\phi = D(\nabla,\mathbf{r})\tilde{\phi}e^{i\psi} = e^{i\psi}D(i(\nabla\psi)+\nabla, \mathbf{r})\tilde{\phi} .$$

Now D can be expanded to first order in $\nabla$ (the WKB approximation) to obtain

$$\nabla\cdot(\tilde{\phi}\frac{\partial D}{\partial \mathbf{k}}) = 0 \text{ or } \frac{d\tilde{\phi}}{ds} + \nabla\cdot\frac{\partial D}{\partial \mathbf{k}} /S = 0,$$

which can be integrated to give the usual WKB amplification factor

$$\tilde{\phi} = \frac{S(0)}{S(s)} .$$

To solve the equations for $\psi$, expressions for $\partial D/\partial\mathbf{k}$ and $\partial D/\partial\mathbf{r}$ must be derived. Once obtained, these expressions must be simplified with a goal of getting an approximate analytic result, or of producing code which can be numerically evaluated efficiently. The explicit r dependence of D can be represented

$$D(\nabla,\mathbf{r}) = D(\nabla,\, a_0(\mathbf{r}), a_1(\mathbf{r}), \ldots, a_m(\mathbf{r}),\, c_0, c_1, \ldots)\ ,$$

where the $a_i$'s are physically convenient parameters such as the imposed magnetic field components or plasma density and the $c_i$'s are constants characterizing the particular situation being studied (e.g. the rf source frequency, or the peak magnetic field amplitude). Let $\mathbf{a} = \{a_0, a_1, \ldots\}$. Then $\partial D/\partial \mathbf{r}$ can be computed using the chain rule for differentiation,

$$\frac{\partial D}{\partial \mathbf{r}} = \frac{d\mathbf{a}}{d\mathbf{r}} \cdot \frac{\partial D}{\partial \mathbf{a}}.$$

Note $d\mathbf{a}/d\mathbf{r}$ is a $3\times m$ matrix which is fixed by the plasma configuration being studied and is not dependent on the plasma model being used as reflected in **K** (this dependence occurs in $\partial D/\partial \mathbf{a}$).

## Implementation of the Ray Calculations on MACSYMA.

The implementation of the calculation of ray trajectories involves the following steps: (1) calculate D in a form where its dependencies on $k_i$ and $a_i$ are explicit; (2) calculate the derivatives $\partial D/\partial \mathbf{k}$ and $\partial D/\partial \mathbf{r}$, then put them in a form suitable for numerical evaluation; (3) use these derivatives in an iterative scheme for solving $d\mathbf{r}/ds$ and $d\mathbf{k}/ds$; and finally (4) present the results graphically. Once the rays have been found, $\tilde{\phi}$ can be computed by evaluating $\mathcal{S}(\mathbf{k},\mathbf{r})$, and $\psi$ by summing $\delta\psi$ along the ray. Finally, a complete solution is obtained by superimposing solutions for the different initial values of **k** and **r** which characterize the source of the excitation. This part of the solution will not be discussed here.

The derivation of D raises two frequently encountered issues. First, the order of the calculation must be considered so that the most simplification can be obtained at each step with a minimum of storage overhead. Second, it is often propitious to make certain approximations on the resulting form of D to avoid unwieldy expressions at later stages (i.e. when computing the derivatives and simplifying the results of differentiation). For the equation of interest here, $D(\mathbf{k},\mathbf{r}) = \mathbf{k} \cdot \mathbf{K}(\mathbf{r}) \cdot \mathbf{k}$, the above concerns motivate us to compute D by expressing **K** as simply as possible,

```
(C2) /* Vector Index of Refraction */
KK : MATRIX( [ KKRR,              %I*KKRT,          -%I*KKRP],
             [ -%I*KKRT,          KKTT,             KKTP    ],
             [%I*KKRP,            KKTP,             KKPP    ])$
```

while retaining its basic symmetry. Once the matrix multiplications have been carried out, and simplifications accomplished (in this case **SCANMAP( MULTTHRU, ... )** suffices) the elements such as KKRR are replaced by expressions such as:

```
(C5) /* Define the remaining elements of KK that are needed. */
KKRR : 1 - WPI2/(1-WCI2)
         - WPI2*AMU/(1-WCI2*AMU^2)$
```

Automatic generation of appropriate type declarations for the temporary variables would make the translation and compilation process less tedious. Finally, as in any such automatic scheme, certain numerical problems may be obscured (like the cancellation of large numbers) or particular restructuring optimizations like Horner's rule may be overlooked. For example, consider the subexpression below:

```
(C14) D10;
                                 2
          2 WCIPHI (1 - WCIPHI ) WPI2          2 WCIPHI WPI2
(D14) - --------------------------- + ----------------------
                   2            2 2              2            2
          (1 - WCIPHI  - WCITHETA )     1 - WCIPHI  - WCITHETA

           3                  2        2                    3
      2 AMU  WCIPHI (1 - AMU  WCIPHI ) WPI2           2 AMU  WCIPHI WPI2
    - ------------------------------------ + ------------------------------
                2          2           2  2              2         2           2
        (1 - AMU  (WCIPHI  + WCITHETA ))       1 - AMU  (WCIPHI  + WCITHETA )
```

This expression results from a straightforward calculation of the derivatives. An obvious optimization can be obtained as shown next:

```
(C15) (E:SUBSTPART(FACTOR(PIECE),%,[1,2]), SUBSTPART(FACTOR(PIECE),E,[2,3]));
                                 2                        5                  2
            2 WCIPHI WCITHETA  WPI2             2 AMU  WCIPHI WCITHETA  WPI2
(D15) - --------------------------- - ------------------------------------------
                       2           2 2              2       2       2         2 2
          (- 1 + WCIPHI  + WCITHETA )    (- 1 + AMU  WCIPHI  + AMU  WCITHETA )
(C16) (E:PART(D15,1,1,1))*MULTTHRU(1/E,D15);
                             2                      1
(D16) 2 WCIPHI WCITHETA  (- -----------------------------
                                        2           2 2
                              (- 1 + WCIPHI  + WCITHETA )

                                                   5
                                                AMU
                           - --------------------------------------) WPI2
                                        2       2       2         2 2
                               (- 1 + AMU  WCIPHI  + AMU  WCITHETA )
```

It is not clear how to apply such optimizations automatically on large expressions. In some cases pattern matching and partial fraction expansions can be applied with some success (this approach was suggested by P. Wang and is currently under investigation). At the time of this work, the OPTIMIZE command was extremely inefficient computationally, but has since been rewritten by M. Genesereth and is now quite fast. Despite some of the drawbacks mentioned above, the use of OPTIMIZE has been very helpful in this application.

The implementation of (3), the iteration scheme for integrating (dr/ds, dk/ds), is achieved by the use of array functions. Array functions have two important advantages over the usual

Here, WPI2 and WCI2 are parameters ($a_i$'s) and AMU is a constant. The number of $a_i$'s might vary between 3 and 10 depending on the plasma model. Approximations can be introduced by expanding in terms of, say, 1/AMU, but for this case it is not necessary.

To accomplish (2) the calculation of derivatives of D, the matrix da/dr is entered (it is usually rather sparse) and multiplied by a list of derivatives obtained by computing $\partial D/\partial a_i$ for each $a_i$. Computing $\partial D/\partial k$ and thus $|\partial D/\partial k|$ is straightforward. Now, it is expected that applying FACTORSUM to various subexpressions may result in a simpler form (note, for instance, the common WPI2 term in KKRR above). This is done by the command

```
SCANMAP( LAMBDA( [X],
                 BLOCK( [Y], Y:FACTORSUM(X),
                             IF Y=X THEN X ELSE Y)),
         .... );
```

where the IF conditional assures the preservation of common subexpressions.

One reason for doing step (2) on MACSYMA is that the matrix arithmetic involves a considerable amount of work if done by hand. But perhaps even more significant is the fact that the MACSYMA command, OPTIMIZE, can now be used to automatically generate a procedure BLOCK for evaluating the expressions efficiently. The BLOCK generated by OPTIMIZE consists of a sequence of assignments of subexpressions to temporary, local variables. For example,

```
(C1) F(A+B^2)+G(A+B^2);
                                 2           2
(D1)                     F(A + B ) + G(A + B )
(C2) OPTIMIZE(%);
                          2
(D2)     BLOCK([T2, T0], T0 : B , T2 : A + T0, RETURN(F(T2) + G(T2)))
```

Using OPTIMIZE is a highly convenient way of accomplishing the familiar programing task of finding common subexpressions, and rewriting the expression in terms of a sequence of statements constituting an evaluation "tree" of the subexpressions. Furthermore, the derivatives for the six equations being integrated (dr/ds and dk/ds) can be calculated in "parallel" (sharing common subexpressions). The BLOCK can be translated and compiled for greater execution efficiency. As might be expected, this optimization often significantly reduces the amount of code required to evaluate an expression, leading to both execution and storage efficiency. A typical list of the derivatives requires 45k words to store on disk (with the SAVE command), and yet the procedure BLOCK generated requires less than 3k words. A more useful comparison would be obtained by writing on disk using FASSAVE (which preserves common subexpressions) or STRINGOUT, but both of these run out of available memory when applied to the original expression.

There are several problems with this method as it is currently implemented. A typical BLOCK might contain a total of 250 temporary variables, when, in fact, a data flow analysis would show that a considerably smaller number of temporaries is needed (i.e. they can be reused).

DO-loop form of specification. First, the order in which particular values of $k_i(s)$ or $r_i(s)$ are computed does not have to be specified. They are computed as needed. This makes it much easier to modify a code since one does not have to be concerned with the order of a sequence of command statements. Second, programs specified this way are highly modular so it is very simple to change one single array function definition in the run time environment, i.e it has both the advantages of a function and of an array. The current liabilities of array functions are: they can use up more storage if used where arrays would not otherwise be used; in the current implementation, references (calls or array accesses) to array functions are not translated or compiled efficiently.

As an example of how the computation of one element of **r** is set up, consider the following MACSYMA commands for implementing a predictor-corrector iteration:

```
(C1) /* Adams-Bashforth Predictor step. */
PSTEP:Y[N-1]+55/24*DY[N-1]-59/24*DY[N-2]+37/24*DY[N-3]-9/24*DY[N-4]$

(C2) /* Corrector step. */
CSTEP: Y[N-1]+9/24*DY[N]+19/24*DY[N-1]-5/24*DY[N-2]+1/24*DY[N-3]$

(C3) /* Example. */
R[N]:=''(SUBST([Y=R,DY=DR],PSTEP));      /* Predict R[N].             */

(D3) R   := R       + 55/24 DR        - 59/24 DR        + 37/24 DR
      N      N - 1              N - 1             N - 2             N - 3

                                                                - 3/8 DR
                                                                        N - 4
(C4)
DR[N]:= ( DSTEP(N),                            /* Computes DR[N] using R[N]. */
          R[N]:''(SUBST([Y=R,DY=DR],CSTEP)), /* Compute corrected R[N].    */
          DSTEP(N),                            /* Compute corrected DR[N].   */
          DR[N]);                              /* Return DR[N].              */

(D6) DR   := (DSTEP(N), R  : 3/8 DR  + R      + 19/24 DR      - 5/24 DR
       N                  N          N    N - 1           N - 1          N - 2

                                                 + 1/24 DR     , DSTEP(N), DR  )
                                                          N - 3              N
```

The function DSTEP computes all the elements of **δr** and **δk** in parallel. Note the ease with which the iteration scheme can be changed. If the array functions were to be compiled, terms like **Y[N]** would be replaced by **ARRAYFUNCALL(Y,N)** in the forms **PSTEP** and **CSTEP**. The derivation of starting points is done separately. In this calculation, each element of **r**, **k**, **δr**, **δk**, and **a** is defined as an array function. While saving elements of **a** is not essential to the integration, it is useful for subsequent calculations to know the trajectory through the parameter space given by **a**(s).

It should be pointed out that in using array functions, one is making a tradeoff between programming convenience versus execution and storage efficiency. To what extent is the inefficiency inherent rather than implementation dependent? The ordinary implementation of array functions in MACSYMA suffers from excessive "number consing" (ref. 4) resulting in a need for large number spaces and costly additional garbage collection. This problem was alleviated by C.

Karney, who implemented a new array function calling routine for the MACSYMA interpreter (not yet installed) which allows LISP number arrays to be used with array functions. The main outstanding difficulty is that array functions cannot be referenced efficiently. In principle however, the check for array elements being undefined should only require one or two machine instructions; thus there is hope that subsequent implementations will have relatively unimportant overhead associated with them.

## Display of the Ray Trajectories.

The graphical display of the ray trajectories employs a rather extensive library package of graphics capabilities implemented by C. Karney, called PLOT2. The main significance of this package is that it interacts with the MACSYMA environment, thus giving both MACSYMA and PLOT2 more power than each would have by themselves. The interactive nature of PLOT2 due to its residing in MACSYMA is particularly advantageous for exploring the parameter space defined by **a**(s). This is done simply by entering a formula depending on the parameters and calling PLOT2 on it. Rescaling and changing view points (in the case of 3-D plots) are very simple interactive operations.

A sample ray trajectory plot is shown in Figure 1. The outer ring is a top view of the torus. The ray starts at the right outside edge of this ring and circles around the torus until it hits the edge again. The inner circle is a projection of the minor crossection of the torus into a single plane. The ray plotting consists of plotting a template indicating the boundaries of the torus and the sector marks followed by calls to PLOT2 using the POLAR option. The template is computed once for each change in aspect ratio and is displayed with REPLOT.

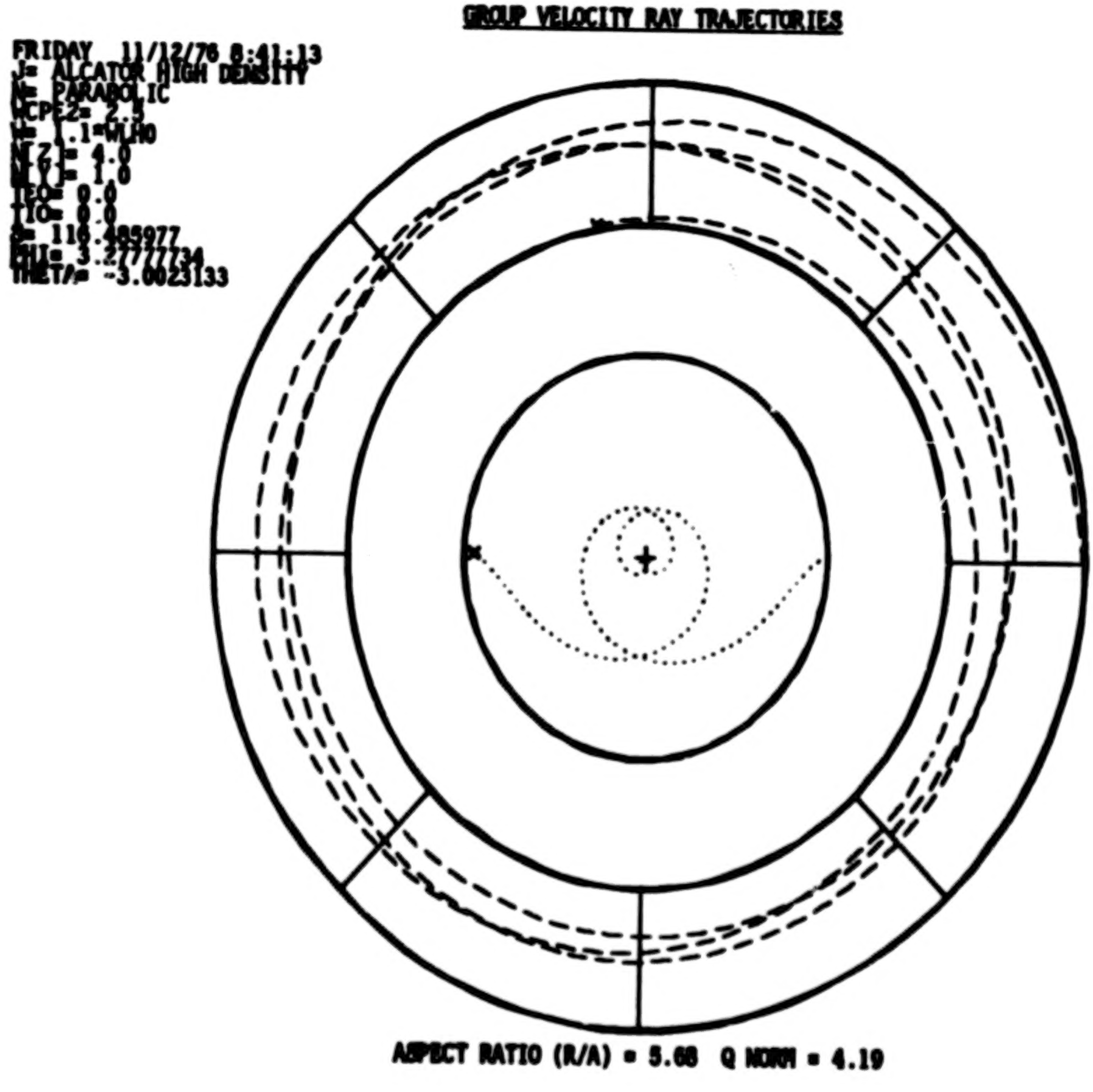
It is important to note that the actual calculation of the rays is invoked by the plotting routine asking for the data in the arrays. Once the array functions and initial conditions have been specified, the array data can be extracted in any order by any other routine without explicitly calling a main program to do the computation. For instance, one may not be directly interested in the rays at all, but simply in the auxiliary parameters, in which case referencing them causes the rays to be computed first.

## Summary.

Space limitations do not permit a more thorough discussion of how the capabilities mentioned here are used in this continuing study. Several different model equations D, and a large number of different parameters are being investigated. The points to be emphasized are: (1) MACSYMA is in some sense evolving into a "complete" system where a user can formulate his equations, approximate and simplify them symbolically, and if need be, study solutions to them numerically and graphically (the drawbacks being that some facilities are not implemented efficiently yet or are too awkward to use); (2) since MACSYMA is a symbolic manipulation environment, it can have facilities to automate various well-defined steps in the creation of numerical procedures; and (3) array functions are an effective way to implement numerical iteration schemes with a degree of simplicity and flexibility uncharacteristic of most numerical programming facilities. A major outstanding problem in generating expressions for numerical evaluation, is finding effective restructuring methods for obtaining expressions which evaluate efficiently (i.e. minimizing multiplications).

## REFERENCES

1. Kulp, J. L.; Johnston, G. L.; and Bers, A.: Progress Report No. 117, Research Laboratory of Electronics, Massachusetts Inst. Tech., 1976, p. 223
2. Roubine, E. ed.: Mathematics Applied to Physics, Springer-Verlag, New York, 1970, p. 286
3. Colestock, P., Bull. Am. Phys. Soc. 21, 1144 (1976)
4. Steele, G.: Fast Arithmetic in MACLISP. Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper No. 22 of this compilation.)


GROUP VELOCITY RAY TRAJECTORIES
FRIDAY 11/12/76 8:41:13
J= ALCATOR HIGH DENSITY
N= PARABOLIC
WCPE2= 2.5
W= 1.1*WLHO
N[Z]= 4.0
N[Y]= 1.0
TEO= 0.0
TIO= 0.0
S= 116.485977
PHI= 3.27777734
THETA= -3.0023133
ASPECT RATIO (R/A) = 5.68 Q NORM = 4.19

BLANK PAGE

BLANK PAGE

# APPLICATION OF MACSYMA TO FIRST ORDER PERTURBATION THEORY IN CELESTIAL MECHANICS*

John D. Anderson and Eunice L. Lau
Jet Propulsion Laboratory

## SUMMARY

The application of MACSYMA to general first order perturbation theory in celestial mechanics is explored. Methods of derivation of small variations in the Keplerian orbital elements are developed. As an example of the methods, the small general relativistic perturbations on the two-body Newtonian motion, resulting from the rotation of the central body, are developed in detail.

## GENERAL PROBLEM

The total acceleration $\ddot{\underline{r}}$ on many objects in the solar system can be written in the following form.

$$\ddot{\underline{r}} = -\frac{\mu \underline{r}}{r^3} + \underline{a}_p \tag{1}$$

where the first term on the right hand side of the expression is the two body acceleration, and the second term is a perturbative acceleration, assumed small enough that a first order perturbation theory is adequate to describe the motion. The zero order solution to equation (1) is the two body solution ($\underline{a}_p = 0$) which yields a Keplerian ellipse with constant orbital elements (a, e, $M_o$, i, $\Omega$, $\omega$).

In this paper we use Gauss's perturbation equations to derive time variations in the Keplerian orbital elements to the first order in the small perturbative acceleration. In terms of radial R, tranverse S, and normal W components of $\underline{a}_p$, the variations in the Keplerian semimajor axis a, the eccentricity e, the mean anomaly $M_o$ at the initial time epoch, the inclination of the orbit i, the longitude of the ascending node $\Omega$, and the argument of the perifocal point $\omega$, are given by the following set of equations (ref. 1).

$$\frac{da}{dt} = \frac{2}{n}(1-e^2)^{-\frac{1}{2}}(Re \sin v + S \frac{p}{r}) \tag{2}$$

---

* The work presented in this paper represents one phase of research carried out at the Jet Propulsion Laboratory, California Institute of Technology, under NASA Contract NAS 7-100.

$$\frac{de}{dt} = \frac{(1-e^2)^{\frac{1}{2}}}{na} \left\{ R \sin v + S \frac{r}{p} \left[ (1 + \frac{p}{r}) \cos v + e \right] \right\} \tag{3}$$

$$\frac{dM_o}{dt} = \frac{r}{na^2 e} \left[ R(\frac{p}{r} \cos v - 2e) - S(1 + \frac{p}{r}) \sin v \right] \tag{4}$$

$$\frac{di}{dt} = \frac{r}{na^2}(1-e^2)^{-\frac{1}{2}} W \cos (v + \omega) \tag{5}$$

$$\sin i \frac{d\Omega}{dt} = \frac{r}{na^2}(1 - e^2)^{-\frac{1}{2}} W \sin (v + \omega) \tag{6}$$

$$\frac{d\omega}{dt} = \frac{r}{na^2 e}(1 - e^2)^{-\frac{1}{2}}\left[-R \frac{p}{r} \cos v + S(1 + \frac{p}{r}) \sin v\right] - \cos i \frac{d\Omega}{dt} \tag{7}$$

where

$$n = (\mu/a^3)^{\frac{1}{2}} \tag{8}$$

$$p = a(1 - e^2) \tag{9}$$

$$r = (\underline{r} \cdot \underline{r})^{\frac{1}{2}} \tag{10}$$

and v is the true anomaly in the polar equation for the Keplerian ellipse.

$$\frac{p}{r} = 1 + e \cos v \tag{11}$$

The application of MACSYMA to the solution of equations (2) through (7) proceeds according to the following steps.

Step 1. Evaluate the components R, S, and W of the perturbative acceleration $\underline{a}_p$.

$$R = \frac{\underline{a}_p \cdot \underline{r}}{r} \tag{12}$$

$$S = \frac{r}{na^2} (1 - e^2)^{-\frac{1}{2}}(\underline{a}_p \cdot \dot{\underline{r}}) - \frac{r}{p} R(e \sin v) \tag{13}$$

$$W = \frac{1}{na^2} (1 - e^2)^{-\frac{1}{2}} \underline{a}_p \cdot (\underline{r} \times \dot{\underline{r}}) \tag{14}$$

The magnitude of the orbital angular momentum $(\underline{r} \times \dot{\underline{r}})$ is $(\mu/p)^{\frac{1}{2}}$, and if $\underline{W}$ is defined as the unit vector normal to the orbital plane along the angular momentum vector, then

$$W = \underline{a}_p \cdot \underline{W} \tag{15}$$

Step 2. Substitute R, S, and W into equations (2) through (7) and simplify.

Step 3. Multiply the six time derivatives from Step 2 by the common factor

$$\frac{dt}{dv} = \frac{r^2}{na^2} (1 - e^2)^{-\frac{1}{2}} \tag{16}$$

Simplify the results to obtain expressions da/dv, de/dv, $dM_o$/dv, di/dv, dΩ/dv, and dω/dv.

Step 4. Integrate the six derivatives from Step 3 between the limits $v_o$ to v. Simplify the results. The resulting six expressions represent the variations Δa, Δe, $\Delta M_o$, Δi, ΔΩ, and Δω as explicit functions of the unperturbed true anomaly v or as implicit functions of time by means of the Keplerian relations between t and v.

Step 5. Obtain the secular time rate of change of the Keplerian elements by evaluating the variations from Step 4 at $v_0 = 0$ and $v = 2\pi$. The rates are given by

$$\dot{a}_s = \frac{n}{2\pi} \Big[\Delta a\Big]_o^{2\pi} \tag{17}$$

with similar expressions for the other elements.

## EXAMPLE

In order to illustrate the general method, we select a relativistic perturbative acceleration that arises because of the rotation of the central body (ref. 2)

$$\underline{a}_p = \frac{6\mu}{c^2} \underline{h} (\underline{r} \cdot \underline{J}) r^{-5} + \frac{2\mu}{c^2} (\dot{\underline{r}} \times \underline{J}) r^{-3} \tag{18}$$

where $\underline{h} = \underline{r} \times \dot{\underline{r}}$ is the orbital angular momentum, and $\underline{J}$ is the spin angular momentum per unit mass for the central body. We choose the equator of the central body as the reference plane for the orientation elements (i, Ω, ω) of the orbit. Then, the spin angular momentum is along the z axis and

$$\underline{J} = (o, o, j) \tag{19}$$

The unit vectors $\underline{P}$, $\underline{Q}$ in the orbit plane, where $\underline{P}$ is directed to perifocus, as well as the vector $\underline{W}$ along $\underline{h}$, are defined by the following MACSYMA statements.

```
(C1)  PX:COS(OMEGA)*COS(NODE)-SIN(OMEGA)*SIN(NODE)*COS(I)$

(C2)  PY:COS(OMEGA)*SIN(NODE)+SIN(OMEGA)*COS(NODE)*COS(I)$

(C3)  PZ:SIN(OMEGA)*SIN(I)$

(C4)  QX:-SIN(OMEGA)*COS(NODE)-COS(OMEGA)*SIN(NODE)*COS(I)$

(C5)  QY:-SIN(OMEGA)*SIN(NODE)+COS(OMEGA)*COS(NODE)*COS(I)$

(C6)  QZ:COS(OMEGA)*SIN(I)$

(C7)  WX:SIN(NODE)*SIN(I)$

(C8)  WY:-COS(NODE)*SIN(I)$

(C9)  WZ:COS(I)$
```

where the Eulerian angles $i$ = I, $\Omega$ = NODE, and $\omega$ = OMEGA are defined in the usual sense.

Now, the position $\underline{r}$ and velocity $\dot{\underline{r}}$ vectors are given by,

$$\underline{r} = x_\omega \underline{P} + y_\omega \underline{Q} \tag{20}$$

$$\dot{\underline{r}} = \dot{x}_\omega \underline{P} + \dot{y}_\omega \underline{Q} \tag{21}$$

where

$$x_\omega = r \cos v \tag{22}$$

$$y_\omega = r \sin v \tag{23}$$

$$\dot{x}_\omega = -(\mu/p)^{\frac{1}{2}} \sin v \tag{24}$$

$$\dot{y}_\omega = (\mu/p)^{\frac{1}{2}} (\cos v + e) \tag{25}$$

The corresponding MACSYMA definitions are as follows:

```
(C30)  XOMEGA:R*COS(V)$

(C31)  YOMEGA:R*SIN(V)$

(C32)  XOMEGADOT:-(M/P)**(1/2)*SIN(V)$

(C33)  YOMEGADOT:((M/P)**(1/2))*(COS(V)+E)$

(C34)  X:XOMEGA*PX+YOMEGA*QX$

(C35)  Y:XOMEGA*PY+YOMEGA*QY$

(C36)  Z:XOMEGA*PZ+YOMEGA*QZ$

(C37)  DX:XOMEGADOT*PX+YOMEGADOT*QX$

(C38)  DY:XOMEGADOT*PY+YOMEGADOT*QY$
```

```
(C39) DZ:XOMEGADOT*PZ+YOMEGADOT*QZ$
```

We now derive expressions for R, S, and W as given by equations (12), (13), and (15) for the perturbative acceleration of equation (18).

First of all, the scalar product of $\underline{r}$ and $(\underline{r} \times \underline{\dot{r}})$ is zero by inspection, so

$$R = (2\mu/c^2)\underline{r} \cdot (\underline{\dot{r}} \times \underline{J})r^{-4} \tag{26}$$

The MACSYMA evaluation of the triple scalar product and then R proceeds as follows:

```
(C40) ENTERMATRIX(3,3);

ROW 1 COLUMN 1  X;

ROW 1 COLUMN 2  Y;

ROW 1 COLUMN 3  Z;

ROW 2 COLUMN 1  DX;

ROW 2 COLUMN 2  DY;

ROW 2 COLUMN 3  DZ;

ROW 3 COLUMN 1  0;

ROW 3 COLUMN 2  0;

ROW 3 COLUMN 3  J;

MATRIX-ENTERED

(C41) DETERMINANT(%);

(C42) RATEXPAND(%);

(C43) RATSUBST(1,SIN(NODE)**2+COS(NODE)**2,%)$

(C44) RATSUBST(1,SIN(OMEGA)**2+COS(OMEGA)**2,%)$

(C45) RATSUBST(1,SIN(I)**2+COS(I)**2,%);

(C46) RATSUBST(1,SIN(V)**2+COS(V)**2,%);

         E COS(I) J SQRT(M) R COS(V) + COS(I) J SQRT(M) R
(D46)    ------------------------------------------------
                             SQRT(P)
    .

    .

    .

(C50) FACTORSUM(D46);

          COS(I) J SQRT(M) R (E COS(V) + 1)
(D50)     ---------------------------------
                      SQRT(P)
```

```
(C51) CAPR:2*M*%/R**4/C**2;

                         3/2
            2 COS(I) J M    (E COS(V) + 1)
(D51)       ------------------------------
                    2
                   C  SQRT(P) R

(C52) RATSUBST(P/R,1+E*COS(V),D51);

(C53) CAPR:%;

                         3/2
            2 COS(I) J M    SQRT(P)
(D53)       -----------------------
                      2  4
                     C  R
```

Because $\underline{a}_p \cdot \dot{\underline{r}} = 0$, the expression for S from equation (13) is obtained as follows:

```
(C55) CAPS:-R*CAPR*E*SIN(V)/P;

                             3/2
              2 E COS(I) J M    SIN(V)
(D55)       - ------------------------
                    2            3
                   C  SQRT(P) R
```

The final component of $\underline{a}_p$, normal to the orbit plane, is obtained by forming the scalar product between $\underline{W}$ and $\underline{a}_p$. First of all we obtain the triple scalar product $\underline{W} \cdot (\dot{\underline{r}} \times \underline{J})$ and then evaluate W with the knowledge from the two body problem that $\underline{W} \cdot \underline{h} = (\mu p)^{\frac{1}{2}}$. The MACSYMA evaluation follows.

```
(C64) ENTERMATRIX(3,3);

ROW 1 COLUMN 1  WX;

ROW 1 COLUMN 2  WY;

ROW 1 COLUMN 3  WZ;

ROW 2 COLUMN 1  DX;

ROW 2 COLUMN 2  DY;

ROW 2 COLUMN 3  DZ;

ROW 3 COLUMN 1  0;

ROW 3 COLUMN 2  0;

ROW 3 COLUMN 3  J;

MATRIX-ENTERED

(C65) DETERMINANT(%)$

(C66) RATSUBST(1,SIN(NODE)**2+COS(NODE)**2,%)$

(C67) RATSUBST(1,SIN(OMEGA)**2+COS(OMEGA)**2,%)$

(C68) RATSUBST(1,SIN(I)**2+COS(I)**2,%)$

(C69) RATSUBST(1,SIN(V)**2+COS(V)**2%);
```

```
   :
   .

(C77) FACTORSUM(D69);

        SIN(I) J SQRT(M) (COS(OMEGA) SIN(V) + SIN(OMEGA) COS(V) + E SIN(OMEGA))
(D77) - -----------------------------------------------------------------------
                                        SQRT(P)
   :
   .

(C82) D77/SIN(I);

(C83) TRIGREDUCE(%);

(C84) %*SIN(I);

(C85) RATSIMP(%);

(C86) FACTORSUM(%);

        SIN(I) J SQRT(M) (SIN(V + OMEGA) + E SIN(OMEGA))
(D86) - ------------------------------------------------
                            SQRT(P)

(C87) %*2*M/C**2/R**3;

(C88) %+6*M*SQRT(M*P)*Z*J*(1+E*COS(V))/C**2/P/R**4;

(C89) FACTORSUM(%);

                      3/2
(D89) -2 SIN(I) J M      (SIN(V+OMEGA) - 3 E COS(OMEGA) COS(V) SIN(V)

                                               2
   - 3 COS(OMEGA) SIN(V) - 3 E SIN(OMEGA) COS   (V) - 3 SIN(OMEGA) COS(V)

                       2            3
   + E SIN(OMEGA))/(C   SQRT(P) R  )

   .
   .
   .

(C91) D89/SIN(I);

(C92) TRIGREDUCE(%);

(C93) FACTORSUM(%)$

(C94) %*SIN(I);

                 3/2
       SIN(I) J M    (3 E SIN(2 V + OMEGA) + 4 SIN(V + OMEGA) + E SIN(OMEGA))
(D94)  -----------------------------------------------------------------------
                                      2           3
                                     C  SQRT(P) R

(C95) CAPW:%$
```

**Now that R, S, and W have been obtained, the variations in the elements can be derived from equations (2) through (7). The MACSYMA expression for de/dt in equation (3) is**

```
(C4)   SQRT(1-E**2)*(CAPR*SIN(V) + CAPS*(R/P)*((1+(P/R))*COS(V)+E))/N/A;
```

We perform some substitutions, and multiply by dt/dv to obtain de/dv as follows.

```
(C5)   RATSUBST(P/A,1-E**2,%);

(C6)   RATSUBST(SQRT(M/A**3),N,%);
```

Now multiply by dt/dv.

```
(C7)   %*(R**2/SQRT(M*P));

(C8)   FACTORSUM(%);

                                 2                                   2        2
(D8)   -2 COS(I) J SQRT(M) (A E R  COS(V) + A E P R COS(V) - P R  + A R

                 2            2 3/2  2
          - A P ) SIN(V)/(A C P     R )


(C9)   RATSUBST(P/(1+E*COS(V)),R,%);

(C10)  RATSUBST(N*A**(3/2),SQRT(M),%);

(C11)  RATSUBST(A*(1-E**2),P,%);

          2 COS(I) J N SIN(V)
(D11)     -------------------
           2              2
          C  SQRT(1 - E )
```

This is the final expression for de/dv.

We will illustrate one more MACSYMA derivation of a variation by determining dΩ/dv from equation (6).

```
(C41)  R*CAPW*SIN(V+OMEGA)/N/A**2/SQRT(1-E**2)/SIN(I);

(C42)  SUBST(SQRT(P/A),SQRT(1-E**2,%);

(C43)  %*(R**2/SQRT(M*P));

(D43)    J M SIN(V + OMEGA) (3 E SIN(2 V + OMEGA) + 4 SIN(V + OMEGA) + E SIN(OMEGA
         -------------------------------------------------------------------------
                                          3/2  2     3/2
                                         A    C  N P


(C44)   J*M/A**(3/2)/C**2/N/P**(3/2);

(C45)   D43/%;

(C46)   TRIGREDUCE(%);

(C47)   FACTORSUM(%);

(D47)   -(4 COS(2 (V + OMEGA)) + 3 E COS(3 V + 2 OMEGA) + E COS(V + 2 OMEGA)

           - 4 E  COS(V) - 4)/2

(C48)   RATSUBST(A*(1-E**2),P,D44);
```

```
(C49)  RATSUBST(N**2*A**3,M,%);

                 J N
(D49)   ------------------------
         2        2  3/2
        C   (1 - E )

(C50)  RATEXPAND(D47);

(C51)  D50/2;

(C52)  RATEXPAND(%);

(C53)  2*%*D49;

                                     3 E COS(3 V + 2 OMEGA)
(D53)  2 J N (-COS(2 (V + OMEGA)) - ----------------------
                                               4

             E COS(V + 2 OMEGA)                      2       2  3/2
           - ------------------ + E COS(V) + 1)/(C (1 - E )    )
                      4
```

This is the final expression for dΩ/dv.

A complete listing of the six derivatives follows.

$$\frac{da}{dv} = 0 \tag{27}$$

$$\frac{de}{dv} = 2n(1 - e^2)^{-\frac{1}{2}} \frac{j}{c^2} \cos i \sin v \tag{28}$$

$$\frac{dM_o}{dv} = \frac{2n}{e} \frac{j}{c^2} \cos i \cos v \tag{29}$$

$$\frac{di}{dv} = 2n(1 - e^2)^{-3/2} \frac{j}{c^2} \left[ 1/2\, e \sin v + 1/2 \sin (v + 2\omega) + \sin (2v + 2\omega) + 3/4 e \sin (3v + 2\omega) \right] \sin i \tag{30}$$

$$\frac{d\Omega}{dv} = 2n(1 - e^2)^{-3/2} \frac{j}{c^2} \left[ 1 + e \cos v - 1/4\, e \cos (v + 2\omega) - \cos (2v + 2\omega) - 3/4\, e \cos (3v + 2\omega) \right] \tag{31}$$

$$\frac{d\omega}{dv} = -2n(1 - e^2)^{-3/2} \frac{j}{c^2} \left[ 3 + \frac{(1 + 2e^2)}{e} \cos v - 1/4\, e \cos(v + 2\omega) - \cos (2v + 2\omega) - 3/4\, e \cos (3v + 2\omega) \right] \cos i \tag{32}$$

MACSYMA has produced expressions which can be integrated by inspection. The secular rates in the elements follow almost immediately.

$$\dot{\omega}_s + \cos i \, \dot{\Omega}_s = - 4n^2(1 - e^2)^{-3/2} \frac{j}{c^2} \cos i$$

$$\sin i \, \dot{\Omega}_s = 2n^2(1 - e^2)^{-3/2} \frac{j}{c^2} \sin i$$

$$(di/dt)_s = 0$$

$$\dot{a}_s = 0$$

$$\dot{e}_s = 0$$

$$\dot{M}_{os} = 0$$

The physical interpretation of these secular expressions is that the perifocal point regresses slowly for satellite motions in the same general direction as the rotation of the central body, and advances slowly for retrograde satellite motions. The line of nodes of the orbit always advances slowly no matter what the value of the inclination angle. The secular variations can be interpreted in terms of a slow dragging of an inertial coordinate system by the rotating central body. This occurs in general relativistic mechanics, but not in Newtonian mechanics where the angular momentum of the central body has no direct effect on the orbital motion. The differences between the two theories of motion are described very well by the example of this paper. The results agree with those obtained by Lense and Thirring (ref. 3).

## REFERENCES

1. Brouwer, D. and Clemence, G. M.: *Methods of Celestial Mechanics*, Academic Press, 1961.

2. Weinberg, S.: *Gravitation and Cosmology*: *Principles and Applications of the General Theory of Relativity*, Wiley, 1972.

3. Lense, Von J. and Thirring, H.: Über den Einflus der Eigenrotation der Zentralkörper auf die Bewegung der Planeten und Monde nach der Einsteinschen Gravitationstheorie. Physik. Zeitschr., vol. XIX, 1918, pp. 156-163.

solution obtained therefore depends on the ability of polynomials to approximate the true solution. A more powerful class of approximating functions is the rational functions. However, the computation of near-minimax rational functions would be best accomplished in the form of Chebyshev-Pade approximations (ref. 11) which require, as an initial step, the generation of Chebyshev series coefficients. Thus the method discussed in this paper is a basic building block as well as a powerful method in its own right.

## 2. CONVERSION TO INTEGRATED FORM

Consider an ordinary differential equation of order $\nu$ with polynomial coefficients:

$$p_\nu(x)\, y^{(\nu)}(x) + \ldots + p_1(x)\, y'(x) + p_0(x)\, y(x) = r(x). \tag{1}$$

We will temporarily ignore the $\nu$ associated conditions which would serve to specify a unique solution of (1). We seek a solution of the form

$$y(x) = \sum_{k=0}^{\infty}{}' \, c_k T_k(x) \tag{2}$$

where the prime (') indicates the standard convention that the first coefficient is to be halved and where $T_k(x)$ denotes the Chebyshev polynomial of the first kind:

$$T_k(x) = \cos\,(k \arccos x).$$

If the series (2) is substituted into the differential equation (1) then the left side of (1) can be expressed in the form of a Chebyshev series. By expressing the right-hand-side polynomial $r(x)$ in Chebyshev form, we can equate coefficients on the left and right to obtain an infinite set of linear equations in the unknowns $c_0, c_1, c_2, \ldots$ (ref. 8). There will be $\nu$ additional equations derived from the associated conditions.) This infinite linear system has the property that the lower triangular part is zero except for a few sub-diagonals and it therefore becomes finite under the assumption $c_k = 0$ ($k >$ kmax), for some chosen kmax. This assumption must be valid, to within some absolute error tolerance, if the solution $y(x)$ is to have a convergent Chebyshev series expansion. Thus one may solve the linear system, for increasing values of kmax, until some convergence criterion has been satisfied.

However, as is noted in reference 8, the linear system is much simpler if (1) is first converted to integrated form. This is because the series resulting from indefinite integration of (2) is much simpler than the series resulting from formal differentiation. Specifically, formal differentiation of (2) yields

$$y'(x) = \sum_{k=0}^{\infty}{}' \left[ \sum_{i=k}^{\infty} 2(2i+1)\, c_{2i+1} \right] T_{2k}(x)$$

$$+ \sum_{k=0}^{\infty} \left[ \sum_{i=k}^{\infty} 2(2i+2)\, c_{2i+2} \right] T_{2k+1}(x) \tag{3}$$

while indefinite integration of (2) yields

$$\int y(x) = \sum_{k=1}^{\infty} (1/2k)\ (c_{k-1} - c_{k+1})\ T_k(x) + K \tag{4}$$

(where K denotes an arbitrary constant). The end result is that in the infinite linear system derived from the integrated form of the differential equation (1), each individual equation contains only a finite number of terms. In the original (differential) form, each individual equation in the infinite linear system is itself infinite. Thus a very substantial reduction in complexity is achieved by considering the integrated form. The coefficients are then specified as the solution of a finite recurrence relation (with non-constant coefficients) rather than an infinite recurrence relation.

The derivation of the recurrence equation is described in detail in the next section. The following theorem gives a formula for the polynomial coefficients of the integrated form of the order $\nu$ differential equation (1), in terms of the polynomials in the original form. This formula for the new polynomials is readily incorporated into a program written in any of the computer languages for symbolic computation, since each new polynomial is specified explicitly as a linear combination of derivatives of the original polynomials (and the new right-hand side is obtained by integrating the original right-hand-side polynomial). An induction proof for Theorem 1 is given in reference 9 and is omitted here.

Theorem 1:

The ordinary differential equation (1) of order $\nu$ with polynomial coefficients $p_\nu(x)$ ,..., $p_0(x)$ and right-hand-side polynomial $r(x)$ is equivalent to the integrated form

$$q_0(x)\ y(x) + \int q_1(x)\ y(x) + \ldots + \int\int \overset{\nu}{\ldots} \int q_\nu(x)\ y(x)$$

$$= s(x) + K_\nu(x) \tag{5}$$

where the polynomial coefficients $q_0(x)$ ,..., $q_\nu(x)$ are given by

# SYMBOLIC COMPUTATION OF RECURRENCE EQUATIONS FOR THE CHEBYSHEV SERIES SOLUTION OF LINEAR ODE'S*

K.O. Geddes
University of Waterloo, Waterloo, Ontario, Canada

## ABSTRACT

If a linear ordinary differential equation with polynomial coefficients is converted into integrated form then the formal substitution of a Chebyshev series leads to recurrence equations defining the Chebyshev coefficients of the solution function. An explicit formula is presented for the polynomial coefficients of the integrated form in terms of the polynomial coefficients of the differential form. The symmetries arising from multiplication and integration of Chebyshev polynomials are exploited in deriving a general recurrence equation from which can be derived all of the linear equations defining the Chebyshev coefficients. Procedures for deriving the general recurrence equation are specified in a precise algorithmic notation suitable for translation into any of the languages for symbolic computation. The method is algebraic and it can therefore be applied to differential equations containing indeterminates.

## 1. INTRODUCTION

The most widely used methods for computing the numerical solution of an ordinary differential equation (ODE), in the form of either an initial-value problem or a boundary-value problem, are discrete-variable methods. That is to say, the solution is obtained in the form of discrete values at selected points. Methods for computing an approximate solution in the form of a continuous function (usually a polynomial or rational function) have received some attention in the literature. Probably the best known continuous-variable method is the Lanczos tau-method (ref. 1) which is closely related to the Chebyshev series methods of Clenshaw (ref. 2) and Fox (ref. 3) for linear ODEs.

---

* This research was supported by the National Research Council of Canada under Grant A8967.

The Chebyshev method has also been used for a first-order non-linear ODE (refs. 4 and 5) but the method then requires iteration whereas it is a direct method in the case of linear ODEs. More recently, the Chebyshev series method has been extended to the solution of parabolic partial differential equations (refs. 6 and 7).

The most extensive treatment of Chebyshev series methods is contained in the book by Fox and Parker (ref. 8). The basic approach is series substitution followed by the solution of resulting recurrence equations. All of the authors treat the series substitution and generation of the recurrence equations as a hand computation prior to the application of a numerical procedure for solving the recurrence equations. However, except for particularly simple special cases, the generation of the recurrence equations is a tedious and error-prone hand manipulation which could well be programmed in a language for symbolic computation. In this paper, procedures are described for generating the recurrence equations for arbitrary-order linear ODEs with polynomial coefficients. There is no need to restrict the method to first and second order equations as previous authors have done. Furthermore, the method can also be applied to problems containing indeterminates (for example, indeterminate initial conditions) and to eigenvalue problems. An attractive feature of the method is that the associated conditions may be of initial-value type, boundary-value type, or any linear combination of function and derivative values at one or more points.

The procedures described here have been implemented in the ALTRAN language (ref. 9). Once the recurrence equations have been generated their solution could, in the standard case, be accomplished by a numerical procedure rather than a symbolic procedure. However, in the potentially powerful application of the method to problems containing indeterminates a symbolic solution of the recurrence equations will sometimes be desired. Therefore this second phase has also been coded in the ALTRAN language. The standard problem without indeterminates is obviously a prime candidate for a hybrid symbolic/numeric computational procedure. In keeping with the potential desire for a symbolic solution, we restrict our attention to a class of problems for which the truncated Chebyshev series can be obtained by a direct method. Thus we consider only linear ODEs with polynomial coefficients. Of course, a linear ODE whose coefficients are rational functions could be converted to one with polynomial coefficients and therefore, in principal, the method can be applied to any linear ODE whose coefficients are functions which can be approximated well by rational functions.

The method assumes that the solution is desired in the interval [-1, 1] (which means that a simple transformation of variables will be required, in general, before applying the method). The truncated Chebyshev series produced by the method is a near-minimax polynomial approximation of the true solution to the problem. This is based on the fact that, for any function continuous in [-1, 1], the minimax error in the truncated Chebyshev series of degree n is never appreciably larger than the error in the best minimax polynomial of degree n (e.g. ref. 10). The goodness of the approximate

$$q_m(x) = \sum_{k=0}^{m} (-1)^{m-k} \binom{\nu-k}{m-k} p^{(m-k)}_{\nu-k}(x), \quad 0 \le m \le \nu \tag{6}$$

and where the right-hand-side polynomial $s(x)$ is given by

$$s(x) = \int\int \overset{\nu}{\dots} \int r(x) \ . \tag{7}$$

In (5) - (7) the notation $K_\nu(x)$ denotes an arbitrary polynomial of degree $\nu-1$ arising from the constants of integration and the notations

$$\int\int \overset{i}{\dots} \int f(x) \quad \text{and} \quad f^{(i)}(x)$$

denote the results of applying, respectively, indefinite integration i times and formal differentiation i times to the function $f(x)$.

## 3. GENERAL FORM OF THE RECURRENCE EQUATION

For an ordinary differential equation of order $\nu$ in the integrated form (5) we seek a solution in the form of the Chebyshev series (2). Substituting (2) into the left side of (5) and removing the summation sign and the $c_k$ outside the integral signs yields

$$\sum_{k=0}^{\infty}{}' c_k \{ q_0(x)T_k(x) + \int q_1(x)T_k(x) + \dots + \int\int \overset{\nu}{\dots} \int q_\nu(x)T_k(x)\} \ . \tag{8}$$

In order to express (8) in the form of a Chebyshev series (where the coefficient of $T_k(x)$ will be a linear combination of $c_i$'s), the polynomials $q_0(x),\dots,q_\nu(x)$ are converted into Chebyshev form. Then the following identities (ref. 8) are applied:

$$T_k(x)T_j(x) = (1/2)\ T_{k+j}(x) + (1/2)\ T_{k-j}(x) \tag{9}$$

$$\int T_i(x) = (1/2(i+1))\ T_{i+1}(x) - (1/2(i-1))\ T_{i-1}(x) \tag{10}$$

where, for the moment, we may assume that k is "large enough" in (9) and that i is "large enough" in (10) to avoid non-positive subscripts. This transforms (8) into the following form, for k large enough (i.e. neglecting the first few terms):

$$\sum_k c_k \{v_0 T_{k+h}(x) + v_1 T_{k+h-1}(x) + \ldots + v_{2h} T_{k-h}(x)\} \tag{11}$$

where the coefficients $v_i$ $(0 \le i \le 2h)$ are rational expressions in k arising from repeated applications of (9) and (10) and h is some positive integer. Then changing the indices of summation in (11), separately in each term, converts (11) into a Chebyshev series of the following form (neglecting the first few terms):

$$\sum_k \{u_0 c_{k-h} + u_1 c_{k-h+1} + \ldots + u_{2h} c_{k+h}\} T_k(x) \tag{12}$$

where the coefficients $u_i$ $(0 \le i \le 2h)$ are rational expressions in k. The first few terms could be derived independently. Finally, by converting the right-hand-side polynomial in (5) into Chebyshev form, we are ready to equate coefficients and solve for the $c_i$'s. The coefficients of $T_0(x), \ldots, T_{\nu-1}(x)$ would not be equated because of the arbitrary term $K_\nu(x)$ appearing in (5). Instead the first $\nu$ equations would come from the associated conditions.

The following example will serve to illustrate. Consider the problem:

$$(1+x^2)\, y''(x) - y'(x) + x\, y(x) = 2-x^2 \tag{13}$$

$$y(0) = 0;\ y'(0) = 1. \tag{14}$$

The integrated form of (13) is, from (5) - (7),

$$\begin{aligned}(1+x^2)\, y(x) + \int (-1-4x)\, y(x) + \iint (2+x)\, y(x) \\ = x^2 - (1/12)\, x^4 + K_2(x).\end{aligned} \tag{15}$$

Substituting (2) into (15) and converting the polynomials into Chebyshev form yields

$$\begin{aligned}\sum_{k=0}^{\infty}{}' c_k \{&[(3/2)T_0(x) + (1/2)\, T_2(x)]\, T_k(x) \\ &+ \int[-T_0(x) - 4T_1(x)]\, T_k(x) \\ &+ \iint[2T_0(x) + T_1(x)]\, T_k(x)\} \\ &= (11/24)\, T_2(x) - (1/96)\, T_4(x) + K_2(x)\end{aligned} \tag{16}$$

where some constant terms on the right have been absorbed into the arbitrary linear term $K_2(x)$. Applying the identities (9) and then (10) yields, after much manipulation, the following form for the factor { } in (16), for k large enough:

$$\begin{aligned}
&\{[8(k+2)(k+3)]^{-1}\,T_{k+3}(x) + (1/4 - (k+2)^{-1} + [2(k+1)(k+2)]^{-1})T_{k+2}(x) \\
&+ (-[2(k+1)]^{-1} - [8(k+1)(k+2)]^{-1})\,T_{k+1}(x) + (3/2 - [(k-1)(k+1)]^{-1})T_k(x) \\
&+ ([2(k-1)]^{-1} - [8(k-1)(k-2)]^{-1})\,T_{k-1}(x) \\
&+ (1/4 + (k-2)^{-1} + [2(k-1)(k-2)]^{-1})T_{k-2}(x) + [8(k-2)(k-3)]^{-1}T_{k-3}(x)\}. \quad (17)
\end{aligned}$$

To obtain the general coefficient of $T_k(x)$ on the left side of (16), the index of summation must be changed separately in each term of the factor (17). For example, for the first term

$$\sum_k c_k\,[8(k+2)(k+3)]^{-1}\,T_{k+3}(x)$$

the desired change of index is $k \leftarrow k-3$, which yields

$$\sum_k\,[8(k-1)k]^{-1}\,c_{k-3}\,T_k(x)$$

where again we are neglecting the first few terms in the series. After changing the indices of summation appropriately, the left side of equation (16) becomes

$$\begin{aligned}
&\sum_k\{[8k(k-1)]^{-1}\,c_{k-3} + (1/4 - 1/k + [2k(k-1)]^{-1})\,c_{k-2} \\
&+ (-[2k]^{-1} - [8k(k+1)]^{-1})\,c_{k-1} + (3/2 - [(k-1)(k+1)]^{-1})c_k \\
&+ ([2k]^{-1} - [8k(k-1)]^{-1})\,c_{k+1} + (1/4 + 1/k + [2k(k+1)]^{-1})c_{k+2} \\
&+ [8k(k+1)]^{-1}\,c_{k+3}\}\,T_k(x). \quad (18)
\end{aligned}$$

Working out the first few terms using special cases (see section 4) of identities (9) and (10), and obtaining the first two equations from the two associated conditions (14), we obtain the following infinite set of linear equations which define the Chebyshev coefficients of the solution function y(x):

$$
\begin{aligned}
&1/2\ c_0 - c_2 + c_4 - c_6 + \dots && = 0 \\
&c_1 - 3c_3 + 5c_5 - 7c_7 + \dots && = 1 \\
&-5/24\ c_1 + 7/6\ c_2 + 3/16\ c_3 + 5/6\ c_4 + 1/48\ c_5 && = 11/24 \qquad (19) \\
&1/48\ c_0 + 0\ c_1 - 17/96\ c_2 + 11/8\ c_3 + 7/48\ c_4 && \\
&\qquad\qquad + 15/24\ c_5 + 1/96\ c_6 && = 0 \\
&1/96\ c_1 + 1/24\ c_2 - 21/160\ c_3 + 43/30\ c_4 + 11/96\ c_5 && \\
&\qquad\qquad + 21/40\ c_6 + 1/160\ c_7 && = -1/96 \\
&\qquad \vdots \qquad\quad \vdots \qquad\quad \vdots &&
\end{aligned}
$$

The remaining equations are obtained by equating to zero the coefficient of $T_k(x)$ in (18), for $k = 5,6,7,\dots$ . Note that (19) is a 7-diagonal system starting from the fourth equation.

In general, the desired Chebyshev coefficients satisfy a (2h+1) - term linear recurrence equation of the form

$$u_0 c_{k-h} + u_1 c_{k-h+1} + \dots + u_{2h}\ c_{k+h} = 0 \qquad (20)$$

where the coefficients $u_i$ are rational expressions in k. Equation (20) will be valid for $k \geq h$ except that the first few right-hand-sides may be nonzero depending on the degree of the right-hand-side polynomial in (5). The value of h depends on the order $\nu$ of the differential equation and on the degree of the left-hand-side polynomials in the integrated form (5). Each application of the product formula (9) and each application of the integration formula (10) increases the value of h by one. Lower and upper bounds on h can be readily determined from the original order-$\nu$ differential equation (1); namely, if maxdeg is the maximum of the degrees of the left-hand-side polynomials in (1) then

$$\nu \leq h \leq \nu + \text{maxdeg}. \qquad (21)$$

The first $\nu$ equations in the infinite linear system come from the associated conditions and will be equations containing an infinite number of terms. If $h > \nu$ then there will follow $\nu$-h "special" cases of the general recurrence equation (20), with nonzero right-hand-sides in general, resulting from equating the coefficients of the terms $T_\nu(x), \ldots, T_{h-1}(x)$. The remaining linear equations result from equating the coefficients of $T_k(x)$, $k = h, h+1, \ldots$ and will all be in the form of recurrence equation (20) except that there will be a few more nonzero right-hand-sides if

$$\deg[s(x)] \geq h,$$

where $s(x)$ is the right-hand-side polynomial in the integrated form (5).

## 4. SPECIAL CASES OF THE RECURRENCE EQUATION

The derivation of the general recurrence equation (20) as described in section 3 is not difficult to implement in a symbolic language. We now consider the derivation of the "special" equations which require the application of modified versions of the product formula (9) and the integration formula (10). In other words, we now want to consider what happens when we drop the assumption that k is "large enough" which was assumed in the derivation of equation (20).

The product formula (9) is in fact correct for all values of k and j if the subscript k-j is replaced by $|k-j|$. The integral formula (10) has a special form for the cases $i = 0$ and $i = 1$, namely

$$\int T_0(x) = T_1(x) \quad \text{and} \quad \int T_1(x) = 1/4\ T_2(x) \tag{22}$$

where an arbitrary constant of integration is implied. These special cases could be incorporated into a program for generating the recurrence equations but the cost of deriving each individual "special" equation would be approximately equal to the cost of deriving the one general equation (20). Fortunately, the form of the special equations can be deduced immediately from the general equation without extra work. Referring to the example in section 3, the third equation of (19) arises from equating coefficients of $T_2(x)$ in the transformed form of (16). If we "blindly" obtain the left-side coefficient of $T_2(x)$ by setting $k=2$ in the general formula (the bracketed expression in (18)) we obtain the equation

$$1/16\ c_{-1} + 0\ c_0 - 13/48\ c_1 + 7/6\ c_2 + 3/16\ c_3 + 5/6\ c_4 + 1/48\ c_5 = 11/24. \tag{23}$$

If the negative subscript is interpreted in absolute value - i.e. if we equate $c_{-1}$ with $c_1$ - then the third equation of (19) results. Our task is now to prove that this "rule" holds in general.

The main point is that negative subscripts may be carried throughout the derivation and their interpretation in absolute value may be postponed until the final step. Theorems 2, 3, and 4 below show that the "special" cases of the recurrence equation can be immediately deduced from the general recurrence equation. Proofs of these theorems appear in reference 9 and are omitted here. The proofs require careful attention to the symmetries involved in the transformations applied to convert (8) into (12).

Theorem 2:

Identities (9) and (10) are valid when non-positive subscripts occur on the left and /or right in the sense that $T_i(x)$ represents $T_{|i|}(x)$.

The following simple example will clarify the application of Theorem 2. Consider the differential equation

$$y'(x) + y(x) = 0$$

or, in integrated form,

$$y(x) + \int y(x) = 0. \tag{24}$$

Substituting the series (2) into (24) yields

$$\sum_{k=0}^{\infty}{}' c_k \{T_k(x) + \int T_k(x)\} = 0. \tag{25}$$

Applying formula (10) gives

$$\sum_{k=0}^{\infty}{}' c_k \{T_k(x) + (1/2(k+1))\, T_{k+1}(x) - (1/2(k-1))\, T_{k-1}(x)\} = 0. \tag{26}$$

The third term in brackets would cause trouble if we evaluated it for k = 1 but we will never do so because we do not equate coefficients of $T_0(x)$. Continuing with the example, the next step is to change indices of summation in (26) yielding

$$\sum_{k=0}^{\infty}{}' c_k T_k(x) + \sum_{k=1}^{\infty}{}' (1/2k)\, c_{k-1}T_k(x) - \sum_{k=-1}^{\infty}{}' (1/2k)\, c_{k+1}T_k(x) = 0. \tag{27}$$

Equating coefficients of $T_k(x)$ on the left and right of (27) gives the general recurrence equation:

$$(1/2k)\ c_{k-1} + c_k - (1/2k)\ c_{k+1} = 0. \tag{28}$$

For this first-order differential equation we must equate coefficients of $T_k(x)$ for $k \geq 1$. Theorem 2 gives a valid interpretation to (26) for each value of the index k but we have yet to prove that (28) is valid when, for example, $k = 1$. In this example, examination of the lower limits of summation in (27) reveals that (28) is clearly valid for $k \geq 2$. The case $k = 0$ will not be required. For $k = 1$ (i.e. equating coefficients of $T_1(x)$), the middle summation in (27) has a factor 1/2 associated with the first term in its sum and the third summation will contribute two terms to the coefficient of $T_1(x)$ - namely, the terms with $k = -1$ and $k = 1$. Thus the coefficient of $T_1(x)$ comes from the terms

$$c_1 T_1(x) + (1/2)(1/2)\ c_0 T_1(x) - (1/2)(1/2(-1))\ c_0 T_{-1}(x) - (1/2)\ c_2 T_1(x).$$

The special form of the recurrence equation corresponding to $k = 1$ should therefore be

$$1/2\ c_0 + c_1 - 1/2\ c_2 = 0. \tag{29}$$

But (29) is precisely the result of setting $k = 1$ in the general recurrence equation (28).

The following two theorems prove that the left side of the general recurrence equation (20) is valid for all $k \geq 1$, in the sense that negative subscripts are to be interpreted in absolute value. Recall that the left side of the general recurrence equation is obtained by transforming (11) into (12). By Theorem 2, the range of the index of summation in (11) may be taken to be 0 to $\infty$ (with the usual "prime" on the summation sign as in (2)). Changing the indices of summation in the terms of (11) transforms (11) into the form

$$\sum_{k=h}^{\infty}{}' v_0(k \leftarrow k-h)\ c_{k-h}\ T_k(x) + \sum_{k=h-1}^{\infty}{}' v_1(k \leftarrow k-h+1)\ c_{k-h+1}\ T_k(x)$$
$$+ \ldots + \sum_{k=-h}^{\infty}{}' v_{2h}(k \leftarrow k+h)\ c_{k+h}\ T_k(x) \tag{30}$$

where the notation $v_i(k \leftarrow f(k))$ denotes, in an obvious way, an operation of substitution in the rational expression $v_i$. Collecting terms, (30) takes the general form (12) where the new rational expressions $u_i$ are given by

$$u_i = v_i(k \leftarrow k-h+i),\ 0 \le i \le 2h. \tag{31}$$

Theorem 3 gives a symmetry property of the rational expressions $v_i$ and then Theorem 4 uses this symmetry property to prove the validity of the general recurrence for $k \ge 1$. In the substitution operations appearing in Theorem 3, the symbol " = " is used in place of the symbol " ← " in order to emphasize the fact that they are arithmetic evaluations in contrast to the change of indices occurring in (30) and (31).

Theorem 3:

The rational expressions $v_i$ $(0 \le i \le 2h)$ appearing in (11) satisfy the following symmetry property:

$$v_i(k=\ell) = v_{2h-i}(k = -\ell),\ 0 \le i \le h$$

for any value of $\ell$.

Theorem 4:

The expression (12), which defines the general form of the recurrence equation, is valid for values of the index $k \ge 1$ in the sense that negative subscripts are to be interpreted in absolute value.

Finally in this section, we mention the interpretation of the term k=0 in (12) which would be required in equating coefficients of $T_0(x)$. Of course for any differential equation (1) of order $\nu \ge 1$ the coefficient of $T_0(x)$ is undetermined because of the constants of integration. However, the method discussed in this paper can be applied directly to a differential equation of order 0:

$$p_0(x)\ y(x) = r(x) \tag{32}$$

in order to compute the Chebyshev series coefficients for an explicit rational function $r(x)/p_0(x)$. In this case the coefficients of $T_k(x)$ on the left and right must be equated for all $k \ge 0$. The coefficient of $T_0(x)$ on the left of the transformed form of (32) is not that obtained by direct application of the general expression in (12):

$$u_0(k=0)\ c_{-h} + u_1(k=0)\ c_{-h+1} + \dots + u_{2h}(k=0)\ c_h. \tag{33}$$

Rather, the correct coefficient of $T_0(x)$ comes from the last h+1 summations in (30) and it is

$$1/2\ v_h(k=0)\ c_0 + v_{h+1}(k=1)\ c_1 + \dots + v_{2h}(k=h)\ c_h. \tag{34}$$

Using (31), (34) becomes

$$1/2\ u_h(k=0)\ c_0 + u_{h+1}(k=0)\ c_1 + \dots + u_{2h}(k=0)\ c_n. \tag{35}$$

Comparing (35) with (33) we see that, for the special case k=0 in (12), the terms with negative subscripts must be ignored and the term in $c_0$ must have a factor 1/2 associated with it.

## 5. SPECIFICATION OF THE PROCEDURES

Procedures for generating the general recurrence equation (20) for the differential equation (1) are specified in a pseudo-Algol algorithmic notation. Four basic "system" functions for polynomial manipulation are assumed:

| | |
|---|---|
| degree (p,x) | - returns the degree of the polynomial p in the indeterminate x |
| derivative (p,x,n) | - returns the n-th derivative of the polynomial p with respect to the indeterminate x |
| coefficient (p,x,n) | - returns the coefficient in the polynomial p of the n-th power of the indeterminate x |
| substitute (r,x,expr) | - returns the result of substituting the expression expr for every occurrence of the indeterminate x in the rational expression r. |

A brief description of each procedure is given followed by the algorithmic specification.

Description of the Procedures:

(1) Procedure generate_recurrence.

Input parameters: $\nu$, p
Output parameters: recurrence_equation, h

The polynomials $p_k$ $(0 \le k \le \nu)$ in the differential equation (1) are passed into the procedure. It is assumed here that the indeterminate in these polynomials is x and it is also assumed that the global array comb has been initialized such that

$$\text{comb}\ (i,j) = \binom{i}{j}.$$

The indeterminate arrays tk and ck are assumed; tk(j) is used to represent the Chebyshev polynomial $T_{k+j}(x)$ where k is an indeterminate and ck(j) is used to represent the term $c_{k+j}$ in the general recurrence equation. k appears only as an indeterminate in these procedures. On return, recurrence_equation is the left side of the general recurrence equation (20) and h is its "half-length" as defined by (20).

Each pass through the m-loop adds one term into factor, where the terms in factor are defined by the bracketed expression in (8). The first part of the m-loop converts the given polynomials into the m-th polynomial of the integrated form, using Theorem 1. Then follow procedure calls which implement the identities (9) and (10). Finally, the appropriate substitutions are performed to transform (11) into (12) which yields the general recurrence equation.

(2) Procedure chebyshev_form.

Input parameters: p, degp
Output: the Chebyshev form of p is returned

The polynomial p of degree degp in the indeterminate x is converted into Chebyshev form. It is assumed that the global array xpower has been initialized such that the element xpower(i) is the Chebyshev form of x**i, using an array of indeterminates t where t(j) represents $T_j(x)$.

(3) Procedure product_tk_times.

Input parameters: p, degp
Output: the representation of $T_k(x)$*p is returned

The polynomial p of degree degp, assumed to be in Chebyshev form, is multiplied by the polynomial $T_k(x)$ by applying identity (9) to each term of p. The indeterminate arrays t and tk are as discussed above.

(4) Procedure integrate.

Input parameters: p, h
Output: the representation of the integral of p is returned

It is assumed that p is a linear combination of the elements tk(-h) ,..., tk(h) where the meaning of the array tk is discussed above. The integral of p is computed by applying identity (10) to each term of p.

```
procedure generate_recurrence ($\nu$,p,recurrence_equation,h)

degp ← degree ($p_\nu$,x)

q ← chebyshev_form ($p_\nu$,degp)

factor ← product_tk_times (q,degp)

h ← degp
```

```
for m=1 step 1 until $\nu$ do
    q ← $p_{\nu-m}$ - ($\nu$-m+1)*derivative($p_{\nu-m+1}$,x,1)
    sign ← -1
    for i=m-2 step -1 until 0 do
        sign ← -sign
        q ← q + sign * comb($\nu$-i,m-i) * derivative($p_{\nu-i}$,x,m-i)
    doend
    degp ← degree (q,x)
    q ← chebyshev_form (q,degp)
    term ← product_tk_times (q,degp)
    hnew ← degp
    for i=1 step 1 until m do
        term ← integrate (term,hnew)
        hnew ← hnew + 1
    doend
    factor ← factor + term
    h ← max (h,hnew)
doend
recurrence_equation ← 0
for j = -h step 1 until h do
    coef ← coefficient (factor, tk(j),1)
    coef ← substitute (coef,k,k-j)
    recurrence_equation ← recurrence_equation + coef * ck(-j)
doend
end of procedure generate_recurrence
```

```
procedure chebyshev_form (p,degp)
   newp ← 0
   for k=0 step 1 until degp do
      newp ← newp + coefficient (p,x,k) * xpower (k)
   doend
   return (newp)
end of procedure chebyshev_form

procedure product_tk_times (p,degp)
   newp ← 0
   for j=0 step 1 until degp do
      newp ← newp + coefficient(p,t(j),1) *(tk(j) + tk(-j))/2
   doend
   return (newp)
end of procedure product_tk_times

procedure integrate (p,h)
   newp ← 0
   for j=-h step 1 until h do
      newp ← newp + coefficient(p,tk(j),1) *(tk(j+1)/(k+j+1)-(tk(j-1)/(k+j-1))/2
   doend
   return (newp)
end of procedure integrate .
```

## 6. SAMPLE PROBLEMS

Reference 9 contains a listing of an ALTRAN program which is an implementation of the procedures in section 5 and also includes an implementation of a method for solving the recurrence equations. The program will accept problems with indeterminates in the associated conditions and also with indeterminates in the differential equation itself. The solution of the recurrence equations is by a method of backward recurrence which obtains a solution under the assumption $c_k=0$ for $k > \text{kmax}$ where kmax is specified. A strategy could easily be implemented for updating kmax until some desired absolute accuracy is satisfied.

The following three sample problems illustrate the application of the method.

Problem 1: (Standard initial-value problem)

$y' = y;\ y(0) = 1$

Value of kmax: 10

Recurrence equation generated:

$$-(1/2k)\ c_{k-1} + c_k + (1/2k)\ c_{k+1} = 0$$

Maximum absolute error in $c_k$ ($0 \le k \le 10$):

$.11\ (10^{-7})$

Size of last computed coefficient:

$c_{10} = .55(10^{-9})$ .

Problem 2: (Complicated boundary-value problem)

$(1+x^2)\ y'' - y' + xy = 2-x^2$

$y(0) = 1;\ y'(0) + 2y(1) - 1/2\ y(-1) = 0$

Value of kmax: 10

Recurrence equation generated:

$$\begin{aligned}&1/8k(k-1)\ c_{k-3} + (1/4 - 1/k + 1/2k(k-1))\ c_{k-2}\\ &- (1/2k + 1/8k(k+1))\ c_{k-1} + (3/2 - 1/(k-1)(k+1))\ c_k\\ &+ (1/2k - 1/8k(k-1))\ c_{k+1} + (1/4 + 1/k + 1/2k(k+1))\ c_{k+2}\\ &+ 1/8k(k+1)\ c_{k+3} = 0\end{aligned}$$

Size of last computed coefficient:

$$c_{10} = .34(10^{-5}).$$

Problem 3: (Indeterminate initial conditions)

$$(1+x^2)\ y'' - y' + xy = 2-x^2$$

$$y(0) = \mu_1;\ y'(0) = \mu_2$$

Value of kmax: 10

Recurrence equation generated: same as problem 2.

Remark: Each $c_k$ is a bilinear polynomial of the form

$$c_k = a_k\mu_1 + b_k\ \mu_2 + d_k\ , \quad \text{for constants } a_k,\ b_k,\ d_k\ .$$

Size of last computed coefficient:

$$c_{10} = .45(10^{-5})\ \mu_1 + .20(10^{-5})\ \mu_2 + .24(10^{-6})\ .$$

Summary of Timing Statistics:

The following table gives the execution times for these three problems on a Honeywell 66/60, where:

$T_1$ = time, in seconds, to generate the general recurrence equation;

$T_2$ = time, in seconds, to solve the equations for $c_k$ $(0 \le k \le 10)$.

| | $T_1$ | $T_2$ |
|---|---|---|
| Problem 1: | 4 | 10 |
| Problem 2: | 160 | 80 |
| Problem 3: | 160 | 73 |

# REFERENCES

1. Lanczos, C.: Applied Analysis. Prentice-Hall, Inc., 1957.

2. Clenshaw, C.W.: The Numerical Solution of Linear Differential Equations in Chebyshev Series. Proc. Cambridge Phil. Soc., vol. 53, pt. 1, Jan. 1957, pp. 134-149.

3. Fox, L.: Chebyshev Methods for Ordinary Differential Equations. Computer J., vol. 4, no. 4, Jan. 1962, pp. 318-331.

4. Norton, H.J.: The Iterative Solution of Nonlinear Ordinary Differential Equations in Chebyshev Series. Computer J., vol. 7, no. 1, Apr. 1964, pp. 76-85.

5. Lewanowicz, S.: Solution of a First-Order Nonlinear Differential Equation in Chebyshev Series. Zastosow. Mat., vol. 15, no. 2, 1976, pp. 251-268.

6. Knibb, D.: The Numerical Solution of Parabolic Partial Differential Equations Using the Method of Lanczos. J. Inst. Math. & Its Appl., vol. 11, no. 2, Apr. 1973, pp. 181-190.

7. Dew, P.M.; and Scraton, R.E.: Chebyshev Methods for the Numerical Solution of Parabolic Partial Differential Equations in Two and Three Space Variables. J. Inst. Math. & Its Appl., vol. 16, no. 1, Aug. 1975, pp. 121-131.

8. Fox, L.; and Parker, I.B.: Chebyshev Polynomials in Numerical Analysis. Oxford Univ. Press, Inc., 1968.

9. Geddes, K.O.: ALTRAN Procedures for the Chebyshev Series Solution of Linear ODE's. Res. Rep. CS-77-05, Univ. Waterloo, Feb. 1977.

10. Powell, M.J.D.: On the Maximum Errors of Polynomial Approximations Defined by Interpolation and By Least Squares Criteria. Computer J., vol. 9, no. 4, Feb. 1967, pp. 404-407.

11. Gragg, W.B.; Johnson, G.D.: The Laurent-Pade Table. Proceedings of the 1974 IFIP Congress, pp. 632-637.

BLANK PAGE

BLANK PAGE

# sin(x)**2 + cos(x)**2 = 1[†]

David R. Stoutemyer
University of Hawaii

## ABSTRACT

This is a chronicle of manifold attempts to achieve tasteful automatic employment of the identities $\sin^2 x + \cos^2 x \equiv 1$ and $\cosh^2 x - \sinh^2 x \equiv 1$, in a manner which truly minimizes the complexity of the resulting expression. After describing the disappointments of trigonometric reduction, trigonometric expansion, pattern matching, Poisson series, and Demoivre's theorem, the author reveals how he achieved his goal by the method of comparative combinatorial substitutions.

## INTRODUCTION

It is no coincidence that the spectre of the identity

$$\sin^2 x + \cos^2 x \equiv 1 \tag{1}$$

is raised in many papers on computer algebraic simplification, such as references 1, 2, and 3. This is a well-known identity, with especially frequent opportunities for employment. The identity

$$i^2 = -1$$

is perhaps the only one that enjoys greater use. However, the former does not share the univariate binomial property of the latter, making a profound difference in the ease of their effective routine use in computer algebra.

Identity (1) and its hyperbolic counterpart

$$\cosh^2 x - \sinh^2 x \equiv 1 \tag{2}$$

are merely the simplest cases of an infinite set of such identities, but I will confine my attention to these two identities because:

1. To my knowledge, none of the existing computer algebra systems provides a totally satisfactory built-in employment of even these two identities.

2. Until these two identities can be treated satisfactorily, why worry about the others.

---

[†] This work was supported by National Science Foundation grant MCS75-22893.

3. In a certain sense, these identities most concisely convey the central facts concerning their constituents: The sine and cosine are dependent, as are their hyperbolic counterparts. The other trigonometric and hyperbolic identities are partly mere reiterations of these facts.

4. I conjecture that dramatic opportunities for these two identities far outnumber those for any other two such trigonometric or hyperbolic identities--perhaps even all of the other such identities combined. Many engineering and science problems utilize sin, cos, sinh, or cosh, raised only to modest powers, with arguments that are mere indeterminates, such as $\theta$, or a product of simple coefficients and indeterminates, such as $\omega t$ or $2\pi x$. For such expressions, application of the few applicable identities other than identities (1) or (2) is most likely to increase the complexity of the expression, as we shall see.

5. A failure to exploit identities (1) or (2) is more noticeable than a failure to exploit more esoteric identities. Uncommitted computer-algebra candidates are quick to notice examples where they can outperform a computer-algebra system. Unfortunately, many who might enjoy and benefit from computer algebra are subject to the all-too-prevalent human tendency to summarily dismiss new opportunities on the basis of a hastily-formed first impression. However, perhaps the scoffer's scorn is somewhat deserved. Is it not embarrassing that computer-algebra systems that can do such an elegant job of factoring and integration cannot exploit one of the few identities that trigonometry students are likely to remember.

I was unconcerned with such matters until I first suffered at the hands of $\sin^2 x + \cos^2 x$. It happened during the testing of a forthcoming more-general tensor version of the vector curvilinear-components function described in reference 4. To make a long story less long, the components of the second-kind Christoffel symbol are computed from those of the contravariant metric tensor and the first-kind Christoffel symbol. These in turn are computed from those of the covariant metric tensor, which are in turn computed from those of the Jacobian matrix, which are computed from the transformation from curvilinear to rectangular Cartesian components. During all of these computations, there are often opportunities to employ identities (1) and (2) when the coordinate transformation involves trigonometric and hyperbolic functions, as do many of the classic orthogonal curvilinear coordinates. Actually, "obligations" is a more appropriate word than "opportunities" here, because if all such opportunities were not exploited as soon as they arose, the computation frequently could not be completed because of storage exhaustion or computing times that had passed the bounds of decency, with no end in sight. The objective was to make the entire computation automatic, untouched by human hands. This objective necessitates a simplifier which exploits one or more instances of identities (1) and (2) in all of their guises, with differing arbitrary subexpressions as the arguments of the trigonometric and hyperbolic functions.

## AN (EDITED) ACCOUNT OF THE AUTHOR'S TRAVAILS

Given a transformation from curvilinear coordinates $\theta_1, \theta_2, \ldots, \theta_n$ to Cartesian coordinates $x_1, x_2, \ldots, x_m$, with $m \geq n$:

$$x_j = f_j(\theta_1, \theta_2, \ldots, \theta_n), \quad (j=1,2,\ldots,m) \ , \tag{3}$$

it was desired to compute the Jacobian matrix **A**, with elements

$$a_{ij} = \frac{\partial f_j}{\partial \theta_i} \quad (i=1,2,\ldots,n) \ . \tag{4}$$

From this, the components of the covariant metric tensor are computed as those of the matrix product

$$\mathbf{G} = \mathbf{A}\,\mathbf{A}^T. \tag{5}$$

The desired second-kind of Christoffel symbol involves linear combinations of the derivatives of **G** and the inverse of **G**, but a general need for automatic trigonometric-hyperbolic simplification was already evident in the results of expression (5) and sometimes even expression (4).

Of the 12 classic orthogonal coordinate systems reported, the coordinate transformations of 8 involve either trigonometric functions, hyperbolic functions, or both. Relatively simple instances of those 8 are

Spherical:

$x = r \sin\theta \cos\phi,$

$y = r \sin\theta \sin\phi,$

$z = r \cos\theta;$

Elliptic Cylindrical:

$x = a \cosh u \cos v,$

$y = a \sinh u \sin v,$

$z = z.$

For orthogonal coordinates, **G** in expression (5) should simplify to a diagonal matrix.

General use of the built-in fractional-power simplifier, RADCAN, was necessary because 2 of the 12 reported coordinate transformations involve square roots and because for vector analysis the square roots of the diagonal elements of **G** are computed.

Using RADCAN alone, it required 2.2 seconds for spherical coordinates and 1.4 seconds for elliptic cylindrical coordinates to compute **G** matrices that were inadequately simplfied. For example, some off-diagonal elements did not simplify to zero in spherical coordinates, and the following values were computed for $\sqrt{g_{11}}$ in spherical and elliptic-cylindrical coordinates respectively:

$$\text{sqrt}(\sin^2\phi + \cos^2\phi)r \sin\theta \ , \tag{6}$$

$$a \ \text{sqrt}(\cosh^2 u \ \sin^2 v + \sinh^2 u \ \cos^2 v) \ . \tag{7}$$

RADCAN alone is clearly inadequate. The lack of off-diagonal zero-recognition had particularly disastrous effects on the computed inverse of G and on the computed Christoffel symbols. Indeed, it often led to storage exhaustion or patience exhaustion during these subsequent calculations.

A perusal of the MACSYMA manual suggests TRIGREDUCE as the obvious candidate for overcoming these problems, and TRIGREDUCE can indeed exploit the syntactically most obvious guises of identities (1) and (2). However, corresponding to expressions (6) and (7), this technique gave

$$\frac{i \ r \ \text{sqrt}[\cos(2\theta) - 1]}{\text{sqrt}(2)} \ , \tag{8}$$

$$\frac{i \ a \ \text{sqrt}[\cos(2v) - \cosh(2u)]}{\text{sqrt}(2)} \ , \tag{9}$$

using 10.4 and 4.5 seconds respectively. Apparently TRIGREDUCE also combines products of trigonometric or hyperbolic functions into corresponding functions of multiple angles, which is more than we want. Other coordinate systems revealed that TRIGREDUCE also combines products of such functions of different arguments into such functions of sums, which is even less desirable in our circumstances.

This suggests following TRIGREDUCE with TRIGEXPAND, to undo these undesired multiple-angles and angle sums. TRIGEXPAND will not expand 1 into $\sin^2\theta + \cos^2\theta$, so we hope for some net simplification from this approximately inverse pair. This pair was followed by RADCAN, for its rational and fractional-power simplification. Although this strategy helped for some coordinate systems, corresponding to expressions (6) and (7) this technique gave

$$\frac{r \ \text{sqrt}(\sin^2\theta - \cos^2\theta + 1)}{\text{sqrt}(2)} \ , \tag{10}$$

$$\frac{a \ \text{sqrt}(\sin^2 v - \cos^2 v + \sinh^2 u + \cosh^2 u)}{\text{sqrt}(2)} \ , \tag{11}$$

using 4.9 and 4.1 seconds respectively. Clearly this strategy is still far from ideal.

Undaunted, I next tried using the pattern matcher as follows:

```
MATCHDECLARE (XTRUE, TRUE) $
TELLSIMPAFTER (SIN(XTRUE)↑2 + COS(XTRUE)↑2, 1) $
TELLSIMPAFTER (COSH(XTRUE)↑2 - SINH(XTRUE)↑2, 1) $
```

This technique failed to simplify some of the spherical-coordinate off-diagonal elements to zero. Also, corresponding to expressions (6) and (7), this technique gave

$$r \sin \theta \tag{12}$$

$$a \text{ sqrt}(\cosh^2 u \sin^2 v + \sinh^2 u \cos^2 v) \tag{13}$$

The patterns are evidently unable to operate together to simplify $\cosh^2 u \sin^2 v + \sinh^2 u \cos^2 v$ to $\sin^2 v + \sinh^2 u$. Also other coordinate systems revealed that the two terms of each pattern are not treated symmetrically. Under the internal ordering, one of each pair is considered to be the "leading variable", which lends a bias towards terms of one type. For example, the expression $\sin^2 x$ is transformed to $1-\cos^2 x$.

More desperate then, I could no longer postpone learning about Poisson series, which are canonical and efficient. In the suite of MACSYMA Poisson functions, OUTOFPOIS seemed more appropriate. However, the fine print revealed some serious restrictions on the allowable arguments of this function. Some, such as the restriction to trigonometric arguments that are linear combinations of indeterminates, with integer coefficients, are fundamental to the nature of Poisson series. Others, such as the limitation to single-precision integers and indeterminates in the trigonometric arguments with names chosen from the set {U,V,W,X,Y,Z}, are concessions to efficiency and ease of implementation. Clearly these restrictions are too severe to permit direct automatic use of OUTOFPOIS from within the curvilinear coordinates function. Nevertheless, if Poisson simplification did the right thing, I was willing to write a front-end filter which feeds OUTOPOIS only those portions of an expression which, with indeterminates temporarily renamed appropriately, meet the restrictions. Although OUTOFPOIS does not perform hyperbolic simplification, I was willing to take what I could get, and I had hopes for using a trick such as replacing cosh $x$ with cos($ix$). However, before investing all of this effort, I tried renaming the coordinate variables manually, then using

```
TRIGSIMP(U):=

   (U:  RATSIMP(U),

   OUTOFPOIS(NUM(U))/OUTOFPOIS(DENOM(U))) $
```

Corresponding to expressions (6) and (7), this technique gave

$$\frac{i \text{ sqrt}(2)\, r \text{ sqrt}[\cos(2u) - 1]}{2}, \tag{14}$$

$$x \text{ sqrt}[(\sinh^2 u - \cosh^2 u) \cos(2v) + \sinh^2 u + \cosh^2 u]/\text{sqrt}(2)\,, \tag{15}$$

using 3.5 and 2.1 seconds respectively. Again we see that the simplification is too drastic, indiscriminately replacing products and powers of trigonometric functions with trigonometric functions of multiple angles and sums. This may be ideal for series approximations to periodic solutions of equations, but it is not ideal for all trigonometric situations. A lovely answer such as $\sin^9 x$, for example, will be converted to an expression truly ugly to behold.

Nevertheless, I think that the above-mentioned front-end filter would be worthwhile in many situations.

At this point, casual perusal of the manual was replaced with an intensive study, which revealed that using EV(..., EXPONENTIALIZE) will convert the trigonometric functions to complex exponentials, which can then be simplified with RADCAN, after which EV(..., DEMOIVRE) converts complex exponentials to sines and cosines. A final RADCAN then gives any spurious "$i$"'s an opportunity to cancel. Corresponding to expressions (6) and (7), this technique gave

$$\begin{aligned} & i\ r\ \mathrm{sqrt}\{[\sin(2\theta) + i\cos(2\theta)]\sin(4\theta) + [\cos(2\theta) \\ & \quad - i\sin(2\theta)]\cos(4\theta) - 2\sin^2(2\theta) - i\sin(2\theta) \\ & \quad - 2\cos^2(2\theta) + \cos(2\theta)\}/2\,, \end{aligned} \tag{16}$$

$$\begin{aligned} & i\ a\ e^{-u}\ \mathrm{sqrt}\{[e^{2u}\sin(2v) + i\ e^{2u}\cos(2v)]\sin(4v) + [e^{2u}\cos(2v) \\ & \quad - i\ e^{2u}\sin(2v)]\cos(4v) + (-e^{4u} - 1)\sin^2(2v) - i\ e^{2u}\sin(2v) \\ & \quad + (-e^{4u} - 1)\cos^2(2v) + e^{2u}\cos(2v)\}/2\,, \end{aligned} \tag{17}$$

using 37 and 16.7 seconds respectively.

The multiple angles were unforeseen, so I tried inserting a TRIGEXPAND between the DEMOIVRE and final RADCAN. Corresponding to expressions (6) and (7), this technique gave

$$\begin{aligned} & r\ \mathrm{sqrt}[\sin^6\theta - 2\ i\cos\theta\sin^5\theta + (\cos^2\theta + 2)\sin^4\theta - 4\ i\cos^3\theta\sin^3\theta \\ & \quad (-\cos^4\theta + 4\cos^2\theta + 1)\sin^2\theta + 2\ i\cos\theta - 2\ i\cos^5\theta\sin\theta - \cos^6\theta \\ & \quad + 2\cos^4\theta - \cos^2\theta]/2\ , \end{aligned} \tag{18}$$

$$\begin{aligned} & a\ e^{-u}\ \mathrm{sqrt}\{e^{2u}\sin^6 v - 2\ i\ e^{2u}\cos v\sin^5 v + (e^{2u}\cos^2 v + e^{4u} + 1)\sin^4 v \\ & \quad - 4\ i\ e^{2u}\cos^3 v\sin^3 v + [-e^{2u}\cos^4 v + (2\ e^{4u} + 2)\cos^2 v + e^{2u}]\sin^2 v \\ & \quad + (2\ i\ e^{2u}\cos v - 2\ i\ e^{2u}\cos^5 v)\sin v - e^{2u}\cos^6 v \\ & \quad + (e^{4u} + 1)\cos^4 v - e^{2u}\cos^2 v\}/2 \end{aligned} \tag{19}$$

using 32.4 and 20.3 seconds respectively.

Now I realize that even if the occurrences of imaginary $i$ all disappeared, this process is much like the Poisson simplification -- too drastic for my purposes.

The MACSYMA primer (reference 5) mentions all of the above techniques, except using REALPART where I used DEMOIVRE, which gives equally disappointing results for this application.

Resolved now to writing my own trig-hyperbolic, simplification function, I first tried the following:

```
TRIGSIMP(U):=
   (U:  RADCAN(U),
   TRIGPOLYSIMP(NUM(U))/TRIGPOLYSIMP(DENOM(U))) $
TRIGPOLYSIMP(U):= BLOCK ([L],
   Make a list L of all unique subexpressions
        which occur as the arguments of both $\sin^m$ and $\cos^n$, with $m,n \geq 2$,
   FOR X IN L DO U:  REMAINDER(U,SIN(X)↑2+COS(X)↑2-1),
   Perform a similar massage for sinh and cosh,
   U) $
```

Corresponding to expressions (6) and (7), this technique gives

$$r \sin \theta \,, \tag{20}$$

$$i\, a \operatorname{sqrt}(\cos v - \cosh u) \operatorname{sqrt}(\cos v + \cosh u) \,, \tag{21}$$

using 2.6 and 3.2 seconds respectively.

Within TRIGPOLYSIMP, using RATSUBST(1,SIN(X)↑2 + COS(X)↑2,U) instead of REMAINDER (U,SIN(X)↑2 + COS(X)↑2-1), and similarly for identity (2) gives virtually identical results.

At the expense of missing opportunities such as replacing $1-\cos^2 x$ by $\sin^2 x$, checking for the presence of both $\sin^m$ and $\cos^n$ removed most of the bias present in the pattern-matching alternative. As revealed by expressions (20) and (21), this technique does an adequate job for these two coordinate systems, though $\sin^2 v + \sinh^2 u$ is slightly preferable to $\cos^2 v - \cosh^2 u$ for computational and esthetic reasons. (I regard "+" as slightly simpler than "-".) This technique also did an adequate job for the other tested coordinate systems, so it is not clear to me now why I looked further. Perhaps it was because I knew that the technique was still too drastic for many purposes. For example, a lovely answer such as $\sin^9 x + \cos^9 x$ is replaced by an expression too obscene to list here.

A way to very nearly retain symmetry and to avoid an increase in expression complexity is to compare the complexities of the expressions obtained by rationally substituting $1-\cos^2 x$ for $\sin^2 x$, by rationally substituting $1-\sin^2 x$ for $\cos^2 x$, and by substituting neither, for each relevant species of $x$ in the expression. Naturally, similar comparisons are done for cosh and sinh. For these comparisons, the least complex candidate wins, with ties broken in an arbitrary asymmetric manner. The complexity function can be designed to reflect the user's value judgements. For simplicity, I defined the complexity as the length of an expression, with the length of a MAPATOM as 1 and the length of a complete subexpression as 1 plus the sum of the lengths of the operands. However, the built-in LISP function ?STRING was a faster length measure, probably because it is a compiled LISP function rather than an interpreted MACSYMA function.

The technique then is to replace the above TRIGPOLYSIMP with a function that makes a set of elements, with each element being, for a unique argument $x$, a set containing sin $x$, cos $x$ or both, according to which of these occur to at least the second power. Analogous elements are also included for cosh and sinh. Then, the appropriate substitutions are successively tried, retaining at each stage the expression with shorter length. Corresponding to expressions (6) and (7), this "comparative sequential substitutions" technique gave

$$r \sin \theta \,, \tag{22}$$

$$a \operatorname{sqrt}(\cosh^2 u \sin^2 v + \sinh^2 u \cos^2 v) \,, \tag{23}$$

using 5.6 and 6.5 seconds respectively. Unfortunately this technique misses opportunities such as replacing $a \cos^2 u \sinh^2 v + a \sin^2 u \cosh^2 v$ by $a(\sin^2 u + \sinh^2 v)$. This example requires replacing $\cos^2 u$ by $1-\sin^2 u$ and $\cosh^2 v$ by $1-\sinh^2$ , but either alone temporarily lengthens the expression, causing the combination to be overlooked.

This phenomenon suggests trying all combinations of feasible substitutions, taking the shortest of these results. Corresponding to expressions (6) and (7), this "comparative combinatorial substitutions" technique gives

$$r \sin \theta \,, \tag{24}$$

$$a \operatorname{sqrt}(\sin^2 v + \sinh^2 u) \,, \tag{25}$$

using 7.1 and 7.6 seconds respectively.

Of course the computing time would grow dramatically with the number of distinct species of sin $x$, cos $x$, sinh $x$, and cosh $x$ that occur to at least the second power, but the computing time grows even more dramatically when less-than-optimally simplified expressions are used for subsequent calculation of the Christoffel symbol components. Also, the combinatorial comparisons are organized in a manner to share some common substitutions between candidates and to eliminate some candidates before computing all of them -- sort of a depth-first substitution and comparison. Moreover, we are dealing with situations where there are not many distinct species. If the combinatorial growth was with respect to the number of terms rather than the number of species, this algorithm would be less practical for this tensor application.

Of the various techniques, I am happiest with this last one of comparative combinatorial substitutions. However, I expect to remain content only until I suffer at the hands of an example such as

$$\text{mess} + 2 \sec^2 x - \tan^2 x \,,$$

which would most esthetically transform to

$$1 + \text{mess} + \sec^2 x \,;$$

or an example such as

$$(\text{mess} + \sin^2 x + \cos^2 x)^{1000} \,,$$

which for most purposes is best replaced by

$$(\text{mess} + 1)^{1000} .$$

Thus, it might be useful to judiciously utilize all 12 trigonometric and hyperbolic functions, together with an inside-out utilization of TRIGPOLYSIMP on all sums, rather than merely the top-level numerator and denominator.

## CONCLUSIONS

I have come to regard identities (1) and (2) as a blessing rather than a curse. The ability to use various judicious combinations of dependent trigonometric and dependent hyperbolic functions often permits a far more compact and understandable answer than is possible when such side relations are not present. The urge to canonicalize in a straightforward fashion can preclude some of these opportunities. It is possible and sometimes necessary to automatically exploit the types of non-canonical simplifications described here.

## ACKNOWLEDGMENTS

I thank David Barton, Richard Bogen, Richard Fateman, Jeffrey Golden, Joel Moses, Stavros Macrakis and Richard Zippel for their assistance during various stages of my quest.

## REFERENCES

1. Fateman, R.J.: Essays in Algebraic Simplification, Harvard Ph.D. Thesis and MIT Project MAC Technical Report 95, 1972.

2. Hearn, A.C.: The Computer Solution of Algebraic Problems by Pattern Matching, University of Utah Computational Physics Report 8, 1971.

3. Moses, J.: Simplification - A Guide for the Perplexed, Communications of the ACM, August 1971.

4. Stoutemyer, David R.: Symbolic Computer Vector Analysis. Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper no. 43 of this compilation.)

5. Mathlab Group: MACSYMA Primer, Project MAC, MIT, 1975.

BLANK PAGE

BLANK PAGE

434

42

# MATRIX COMPUTATIONS IN MACSYMA

Paul S. Wang
Laboratory for Computer Science & Mathematics Department, MIT*

## INTRODUCTION

An important facility for a computer symbolic mathematics system is matrix computation. MACSYMA provides many built-in facilities for manipulating matrices. The matrices may have numerical or symbolic entries. This means matrix elements may involve indeterminates and functional expressions. Computations will be done exactly, keeping symbols as symbols. The purpose of this article is to describe these matrix facilities, to explain their use and to give some idea as to the algorithms or procedures used.

In section 2 the question of how to form a matrix and how to create other matrices by transforming existing matrices within MACSYMA is addressed. Arithmetic and other computation with matrices is discussed in section 3. The user control of computational processes through the use of OPTION VARIABLES is indicated in section 4. In sections 5 and 6 two algorithms designed specially for sparse matrices are given. Section 7 compares the computing times of several different ways to compute the determinant of a matrix.

## FORMING AND TRANSFORMING MATRICES

Matrices are created in MACSYMA by entering a new matrix, making transformation on an existing matrix or collecting elements of an array or coefficients of a set of linear equations. To enter a matrix, for example,

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

one just types MATRIX ([A,B], [C,D]). If the matrix is large and one wishes to type the entries one at a time then the command A:ENTER(m,n); can be used. The integers m and n are the dimensions of the matrix to be entered. Sometimes the value of an entry can be expressed as a function of the row and column indexes. In this case the command GENMATRIX which generates a matrix from a MACSYMA array is useful. For instance, if an m x n matrix A is needed with $A_{ij} = i/j$, one first defines an array B by B[I,J]:= I/J. Then the command

---

*This work was supported by ERDA contract E11-1-3070 and by NASA grant NSG 1323.

GENMATRIX(B,m,n) will construct the desired matrix. The MACSYMA reference manual (ref. 1) contains a more detailed description of GENMATRIX.

The command A:INDENT(m) produces an m x m identity matrix; A:DIAGMATRIX(m,*x*) produces an m x m diagonal matrix with each diagonal entry *x*.

MACSYMA provides several commands for taking a part or a submatrix of an existing matrix. The command MINOR(A,i,j) produces a new matrix by deleting row i and column j from A. ROW(A,n) and COL(A,n) give, as a matrix, the *nth* row and column of A respectively. In general, SUBMATRIX($i_1$, ... , $i_m$, A, $j_1$, ... ,$j_n$) produces a matrix from A by deleting the rows $i_1$, ... $i_m$ and columns $j_1$, ... ,$j_n$. The (i,j)*th* entry in a matrix A is accessed by typing A[i,j].

There are also facilities for modifying or transforming a given matrix. TRANSPOSE (A) returns $A^T$. ADDROW(A,R) produces a matrix which is equal to A with R appended as the last row. MATRIXMAP(fn,A) creates a new matrix of the same dimensions as A where each entry is formed by applying the given function fn to each element of A. The function fn can be a MACSYMA function or a user defined function. For example, if one wants to make a matrix of numerators of the entries of A one can do MATRIXMAP(NUM,A).

A user can change the (i,j)*th* entry of a matrix A, to x, say, by typing A[i,j]:x. This change is made <u>on A</u>. If one wishes a new matrix then the change should be made on a copy of A. COPYMATRIX(A) gives a new matrix which is a copy of A.

As a rule, MACSYMA commands will not alter existing expressions. There are a few exceptions to this rule and they are clearly indicated in the MACSYMA reference manual (ref. 1). To emphasize the effect of an expression-altering command we show the following example:

```
(C1)  A:MATRIX([1,2], [4,7])$


(C2)  A: [2,2]: 9$


(C3)  A;
```

$$\begin{pmatrix} 1 & 2 \\ 4 & 9 \end{pmatrix}$$

Let a set of linear equations EQ1, ... , EQm in the variables X1, ..., Xm be given. The commands

COEFMATRIX(eqlist,varlist) and AUGCOEFMATRIX(eqlist,varlist) are used to produce the coefficient matrix and augmented coefficient matrix, respectively, where eqlist is [EQ1, ..., EQm] and varlist is [X1, ... ,Xm].

## MATRIX COMPUTATIONS

Between two matrices of the same dimension and between a scalar and a matrix the arithmetic operators +, -, *, ↑ and / are used for an elementwise effect. Thus if

$$\begin{pmatrix} x & y \\ z & w \end{pmatrix}$$

then

$$A\uparrow{-1} = 1/A = \begin{pmatrix} 1/x & 1/y \\ 1/z & 1/w \end{pmatrix}$$

and

$$A\uparrow 2 = A*A = \begin{pmatrix} x^2 & y^2 \\ z^2 & w^2 \end{pmatrix}$$

The usual matrix multiplication uses the dot operator. Multiplying a matrix by itself a number of times is indicated by the operator ↑↑. Thus

$$A\uparrow\uparrow 2 = A.A = \begin{pmatrix} x^2+yz & xy+yw \\ zx+zw & zy+w^2 \end{pmatrix}$$

As an aside, we should note that these operations are not exclusively reserved for matrices: the dot and ↑↑ operators are used for noncommutative multiplication and powers in general. Computation involving noncommutative multiplication between variables can be done by declaring the variable NONSCALAR and using the dot operator. For example:

```
(C1) DECLARE ([A, B], NONSCALAR)$

(C2) (A + B) . (A - B), EXPAND;
```

$$A^{<2>} + B.A - A.B - B^{<2>}$$

Note how exponents resulting from noncommutative multiplication are displayed. The inverse of A is A↑↑-1. Among these matrix computations, the inverse is the most time consuming. The exact inverse of a matrix whose entries are polynomials, rational functions and other functional expressions is often much larger than the matrix itself. In some cases, moderately-sized symbolic matrices (under 10 X 10, say) with not very complicated entries may have inverses whose size exceeds the maximum store available to MACSYMA. In other cases, the inverse is of reasonable size but the computation runs out of store at an intermediate stage. This difficulty, called intermediate expression swell, is common to many other symbolic computation processes: polynomial greatest-common-divisor calculation (GCD), factoring and definite integration, just to name a few. The challenge to algorithm designers is to avoid or control intermediate expression growth while keeping the algorithms reasonably fast. In general, the best procedure to use is dependent on the problem to be solved. There are two different inversion procedures in MACSYMA: a basic Bareiss-type Fraction Free Gaussian Elimination (FFGE) algorithm (ref. 2) and a special procedure for sparse matrices. The latter is a special feature in MACSYMA and will be described in the section, "Inverse of Sparse Matrices."

The FFEG uses the usual Gaussian elimination process which reduces the given matrix to the identity by elementary row operations while transforming an identity matrix appended to the given matrix to the desired inverse. However, in order to avoid computing with fractional forms which involves many costly GCD calculations, the elimination is made fraction-free. First each row is multiplied by the least common multiple of its denominators. Then the elimination is carried out with cross multiplication instead of division. Significant improvement in speed results from fraction-free elimination. However, cross-multiplication adds to intermediate expression growth.

When the FFGE has reduced a given matrix to upper triangular form, the last diagonal element is equal to the determinant of the (rescaled) matrix. Therefore it is also a method for computing the determinant of a matrix. The command in MACSYMA using this technique to calculate a determinant is DETERMINANT(A). There are three other ways to compute the determinant also implemented in MACSYMA. These will be described in section 6. One can also obtain the triangular form, the echelon form (essentially the triangular form with the first entry of each row normalized to 1), the rank and characteristic polynomial of a matrix A by TRIANGULARIZE(A), ECHELON(A), RANK(A), and CHARPOLY(A,x), respectively.

## OPTIONS IN CONTROLLING COMPUTATION

Matrix computations can result in expressions which are rather large and complicated. Therefore it is important to carefully control the manner in which a given computation is executed. User control options are provided in MACSYMA in the form of OPTION VARIABLE or SWITCH settings. There are many SWITCHES in MACSYMA. Each SWITCH may have two or more possible settings which affect the behavior of one or several routines controlled by the SWITCH. A SWITCH is set, like any other variable, by using the : operator. For example, if RATMX:TRUE is done, then all matrix arithmetic will be done in CRE form (ref 1). In a fresh MACSYMA system, each SWITCH has a default value or setting. RATMX has the default value FALSE, which means MATRIX arithmetic will be done in general representation. Vectors in MACSYMA can be represented as one-dimensional matrices. However it is often convenient to represent vectors as lists. A list V:[A, B, C] represents a row vector. To mix computation with lists and matrices one sets LISTARITH to TRUE. If A is a 3 X 3 matrix then V.A is a 1 X 3 matrix and A.V is a 3 X 1 matrix. Setting SPARSE to TRUE enables several routines specially designed for sparse symbolic matrix computations to be activated. Other options control operations of scalar-matrix arithmetic and noncommutative operations. The available options are described in detail in the manual. Efficient use of these controls comes with experience with a given application, and experimentation.

## INVERSE OF SPARSE MATRICES

The question of whether the inverse of a given matrix will fit in the available memory space to MACSYMA depends on the size, the number of indeterminates and the number of zero entries in the matrix. A matrix with many zero entries is said to be sparse. Sparse matrices occur frequently in practice. One often-asked question in connection with inverting a sparse matrix is how to order the rows and columns to facilitate the computation. MACSYMA has programs for reordering rows and columns. We present its algorithm here in more detail to provide the user with a deeper insight.

If the given matrix is sparse its inverse may also have many zero entries. One obvious example of this situation is a triangular matrix. Substantial computation can be saved if the zero entries in the inverse are predicted so that they do not have to be computed. It has been shown that this can be done if and only if the given matrix is block reducible (ref. 3). Let Q be an n x n sparse matrix. If there is a way of reordering rows and columns so that Q becomes

$$\hat{Q} = \begin{pmatrix} \hat{Q}_{11} & \hat{Q}_{12} & \hat{Q}_{13} & \cdots & \hat{Q}_{1t} \\ 0 & \hat{Q}_{22} & \hat{Q}_{23} & \cdots & \hat{Q}_{2t} \\ 0 & 0 & \hat{Q}_{33} & \cdots & \hat{Q}_{3t} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \hat{Q}_{tt} \end{pmatrix} , \quad t \geq 1,$$

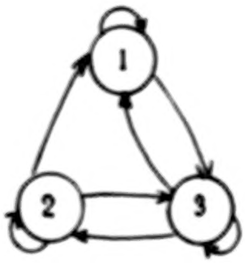
where $\hat{Q}_{ij}$ is a matrix of dimension $n_i \times n_j$, $n_1 + \ldots + n_t = n$ and if $t > 1$ then Q is reducible. Otherwise Q is irreducible. A fairly efficient algorithm is implemented in MACSYMA for computing $\hat{Q}^{-1}$ from $\hat{Q}_{ii}^{-1}$. $\hat{Q}^{-1}$ has the same block structure as $\hat{Q}$. To obtain $Q^{-1}$ from $\hat{Q}^{-1}$ is just a matter of undoing the row and column permutations that transformed Q to $\hat{Q}$.

Now let us consider the means of obtaining the desired block structure. A directed graph (ref. 3) g(Q) can be associated to the matrix Q. This graph has n nodes labeled 1 through n. The nodes are linked by directed edges representing nonzero entries of Q. An edge from node i to node j represents the nonzero entry $q_{ij}$. This edge is labelled $q_{ij}$. Only the nonzero entries of Q are represented in g(Q). A sequence of edges leading from node i to j is called a path from i to j. A subgraph is isolated if any pair of nodes in the subgraph are connected and no nodes outside the subgraph are connected to any inside; such isolated subgraphs are called strong components of g(Q). The strong components of g(Q) give rise to the block structure of Q. We denote by $S_Q$ the number of strong components in g(Q).

The outcome of the above scheme is dependent on the given order of the rows and columns of Q. This means that a permutation of the rows and/or columns may result in an associated graph with more strong components and therefore lead to a refined block structure of Q. For example, if Q is given as

$$Q = \begin{pmatrix} 0 & 0 & a \\ b & c & d \\ e & f & g \end{pmatrix}$$

then g(Q) looks like



which has only one strong component. However, by interchanging the first and third rows of Q one would find $t = 2$. Indeed two is the maximum number of blocks Q has. As a matter of fact Q can always be fully reduced if nonzero elements are assigned on the main diagonal before constructing g(Q).

## DETERMINANT OF SPARSE MATRICES

There are four different ways to compute a determinant in MACSYMA. If RATMX is FALSE the DETERMINANT command uses general representation and a Bottom-Up minor expansion (BU) suggested by Gentleman and Johnson (ref. 4). The BU method computes all possible 2 x 2 minors in the last two columns (rows). Then all the 3 x 3 minors, etc. The BU method was also programmed in LISP by Fateman to render expressions in CRE form. The command using it is NEWDET. If RATMX is TRUE, then one of two methods is used by the DETERMINANT command depending on the setting of SPARSE. If SPARSE is FALSE, the FFGE method mentioned before is used. If SPARSE is TRUE, then a routine, TDBU, specially designed for taking the determinant of matrices with many zero entries is called.

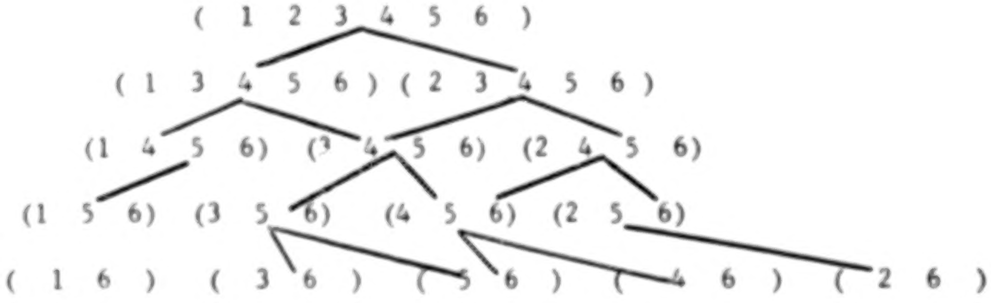
We describe the TDBU sparse determinant algorithm in more detail, since we believe it to be one of the most efficient methods for this purpose currently implemented on a symbolic mathematical computer system.

If the given matrix, Q, is reducible to a block triangular form, then its determinant is the product of the determinants on the main diagonal multiplied by 1 or -1, depending on the row-column reordering. Let us assume Q is sparse and irreducible. A minor expansion method is employed for the determinant of Q. It consists of a Top-Down analysis phase and a Bottom-Up computation phase. The Top-Down phase constructs a graphical structure of minors needed to be computed and the interdependence between these minors. This avoids almost all unnecessary minors. Then the minors needed are computed Bottom-Up so that there is no repeated computation. The method is named TDBU (ref. 5).

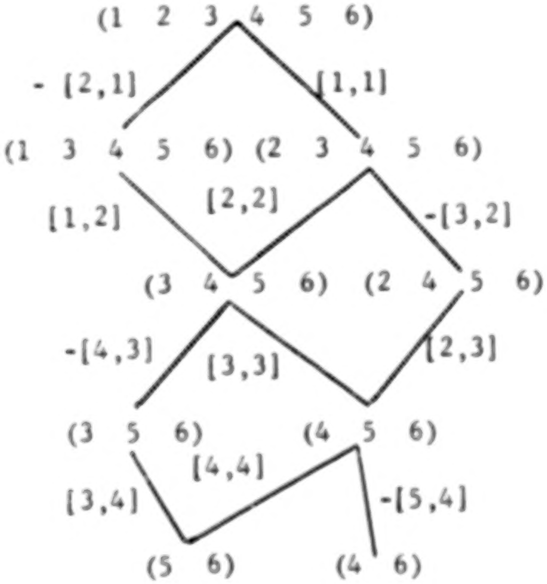
Let us illustrate the TDBU by an example. Consider the 6 x 6 tridiagonal matrix.

$$Q = \begin{pmatrix} B & C & 0 & 0 & 0 & 0 \\ A & B & C & 0 & 0 & 0 \\ 0 & A & B & C & 0 & 0 \\ 0 & 0 & A & B & C & 0 \\ 0 & 0 & 0 & A & B & C \\ 0 & 0 & 0 & 0 & A & B \end{pmatrix}$$

By the list $(i_1, ..., i_k)$ we denote the minor at the intersection of the last k columns and the rows $i_1, i_2, ..., i_k$. Using the position of the nonzero entries the following tree is constructed:


( 1 2 3 4 5 6 )
( 1 3 4 5 6 )
( 2 3 4 5 6 )
(1 4 5 6)
(3 4 5 6)
(2 4 5 6)
(1 5 6)
(3 5 6)
(4 5 6)
(2 5 6)
( 1 6 )
( 3 6 )
( 5 6 )
( 4 6 )
( 2 6 )

There are 14 nodes besides the root. However some of these nodes represent obviously singular minors. If a singularity check is used which looks for an entire row or column of zeros in a minor, several branches can be cut from this tree. With signed multiplier labels attached to the branches the tree structure now becomes the following:



Therefore, only 8 minors need be computed. As the bottom-up computation progresses minors no longer needed are discarded. Thus the storage required for minors is limited to slightly more than one set of necessary i x i minors.

## TIMING COMPARISONS

Timing tests have been conducted for the three different methods for determinant computations: the fraction-free Gaussian elimination (FFGE), the bottom-up minor expansion (BU), and the TDBU. Two forms of sparse matrices are used: the tridiagonal (TRID) and the tridiagonal with a block structure (BLK). In the following tables an X indicates running out of core. The timings (including garbage collection time) are measured on a DEC KL-10.

```
                         BLK(6)

          [ C + B  C + A    0      0      0      0   ]
          [                                          ]
          [ B + A  C + B  C + A    0      0      0   ]
          [                                          ]
          [   0      0    C + B  C + A    0      0   ]
          [                                          ]
          [   0      0    B + A  C + B  C + A    0   ]
          [                                          ]
          [   0      0      0      0    C + B  C + A ]
          [                                          ]
          [   0      0      0      0    B + A  C + B ]
```

| BLK DIMENSION | FFGE | BU | TDBU |
|---|---|---|---|
| 6 | 1405 | 166 | 209 |
| 8 | 5310 | 684 | 356 |
| 10 | 17410 | 1523 | 863 |
| 12 | 43883 | 2952 | 1163 |
| 14 | 104642 | 5933 | 1584 |
| 16 | X | 16763 | 2044 |
| 18 | X | X | 3006 |
| 20 | X | X | 3643 |
| 22 | X | X | 4807 |
| 24 | X | X | 6107 |
| 26 | X | X | 7992 |
| 28 | X | X | 9507 |

time in milliseconds

```
                TRID(6)

[ C + B  C + A    0      0      0      0   ]
[                                          ]
[ B + A  C + B  C + A    0      0      0   ]
[                                          ]
[   0    B + A  C + B  C + A    0      0   ]
[                                          ]
[   0      0    B + A  C + B  C + A    0   ]
[                                          ]
[   0      0      0    B + A  C + B  C + A ]
[                                          ]
[   0      0      0      0    B + A  C + B ]
```

| TRID<br>DIMENSION | FFGE | BU | TDBU |
|---|---|---|---|
| 6 | 1563 | 177 | 214 |
| 8 | 6949 | 710 | 758 |
| 10 | 22750 | 1281 | 1446 |
| 12 | 57251 | 2760 | 2114 |
| 14 | 140445 | 6099 | 2970 |
| 16 | X | 17307 | 4739 |
| 18 | X | X | 6921 |
| 20 | X | X | 9367 |
| 22 | X | X | 12565 |
| 24 | X | X | 17132 |
| 26 | X | X | 23138 |
| 28 | X | X | 30319 |

time in milliseconds

## REFERENCES

1. Mathlab Group: MACSYMA Reference Manual. Lab. Compu. Sci., Massachusetts Inst. Technol., Nov. 1975.

2. Lipson, J.D.: Symbolic Methods for the Computer Solution of Linear Equations With Applications to Flow-Graphs. Proceedings of the 1968 Summer Institute on Symbolic Mathematical Computation, IBM Corp., 1969, pp. 233-303.

3. Wang, P.; and Minamikawa, T.: Taking Advantage of Zero Entries in the Exact Inverse of Sparse Matrices. Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation, Aug. 1976, pp. 346-350.

4. Gentleman, W. M.; and Johnson, S. C.: Analysis of Algorithms, A Case Study: Determinants of Matrices With Polynomial Entries. ACM Trans. Math. Software, vol. 2, no 3, Sept. 1976, pp. 232-241.

5. Wang, P.: On the Expansion of Sparse Symbolic Determinants. Proceedings of the 10th Hawaii International Conference on System Sciences, Jan. 1977.

# SYMBOLIC COMPUTER VECTOR ANALYSIS*

David R. Stoutemyer
University of Hawaii

## ABSTRACT

A MACSYMA program is described which performs symbolic vector algebra and vector calculus. The program can combine and simplify symbolic expressions including dot products and cross products, together with the gradient, divergence, curl, and Laplacian operators. The distribution of these operators over sums or products is under user control, as are various other expansions, including expansion into components in any specific orthogonal coordinate system. There is also a capability for deriving the scalar or vector potential of a vector field. Examples include derivation of the partial differential equations describing fluid flow and magnetohydrodynamics, for 12 different classic orthogonal curvilinear coordinate systems.

## INTRODUCTION

Vector algebra and vector calculus enjoy diverse use throughout engineering, science, and mathematics. Vector analysis lends conciseness that often simplifies the derivation of mathematical theorems and the statement of physical laws. Vector notation often clearly conveys geometric or physical interpretations that greatly facilitate understanding. At one extreme, vector analysis provides a systematic method for deriving the mathematical statement of physical laws in specific orthogonal curvilinear coordinate systems. At another extreme, vector analysis provides a means of stating and operating on these physical laws independent of a coordinate system, free from the distracting details of individual components.

However, many engineers and scientists do not use vector analysis frequently enough to remain familiar with many of the special vector identities that are sometimes crucial to simplifying vector expressions. Also, though systematic, the expansion of vector expressions into specific orthogonal curvilinear components is usually tedious and fraught with opportunities for blunders. Other tedious blunder-prone operations include deriving scalar or vector potentials from given vector fields. This article describes a computer program which helps overcome these human frailties by automating these processes.

---

*This work was supported by National Science Foundation grant MCS75-22893.

The next section gives a brief demonstration of the program. Subsequent sections outline the underlying mathematical and programming techniques, then summarize the performance for more comprehensive examples.

## A DEMONSTRATION

The vector-analysis package contains various default and optional simplifications for the dot and cross products together with the operators, GRAD, DIV, CURL, and LAPLACIAN. The vector operands may be an arbitrary mixture of similar-length ordered lists, representing the specific components, together with indeterminates declared NONSCALAR, representing the vectors as abstract entities. For example, to establish P, Q, F and G as vector entities, we type

```
(C3) DECLARE([P, Q, F, G], NONSCALAR) $
```

Now, let's attempt to prove the following vector identity, where "~" represents the cross product operator:

```
(C4) (P~Q).(F~G) + (Q~F).(P~G) + (F~P).(Q~G) = 0;
(D4)    F • (G ~ Q) ~ P - F • (G ~ P) ~ Q + F • G ~ P ~ Q = 0
```

Evidently the default simplifications are not drastic enough, so we type

```
(C5) VECTORSIMP(%), EXPANDALL;
(D5)                             0 ≐ 0
```

Now, let's determine the expansion of an expression involving vector differential operators:

```
(C6) EXAMPLE:   LAPLACIAN(%PI*(S+H)) = DIV(3*S*P);
                      %PI LAPLACIAN (S + H) = 3 DIV (P S)

(C7) VECTORSIMP(EXAMPLE), EXPANDALL;
(D7) %PI LAPLACIAN S + %PI LAPLACIAN H = 3 DIV P S + 3 P • GRAD S
```

Suppose that we wish to find the specific representation of this equation in parabolic coordinates. To avoid having to look up the definition of parabolic coordinates:

```
(C9) BATCH(COORDS);

(C10) TTYOFF:TRUE $

(C13) /* PREDEFINED COORDINATE TRANSFORMATIONS:
CARTESIAN2D, CARTESIAN3D,
POLAR, POLARCYLINDRICAL,
SPHERICAL, OBLATESPHEROIDAL, PROLATESPHEROIDAL,
   OBLATESPHEROIDALSQRT, PROLATESPHEROIDALSQRT,
ELLIPTIC, ELLIPTICCYLINDRICAL, CONFOCALELLIPTIC,
   CONFOCALELLIPSOIDAL,
PARABOLIC, PARABOLICCYLINDRICAL, PARABOLOIDAL,
BIPOLAR, BIPOLARCYLINDRICAL,
TOROIDAL,
CONICAL */

/* RESERVED COORDINATE VARIABLES AND PARAMETERS: */

LISTOFVARS(COORDS);
(D13)          [X, Y, Z, R, THETA, PHI, E, U, V, F, W, G]

(D14)                           BATCH DONE
```

In general, coordinates are specified as a list with the first element being a list of the transformation to a set of rectangular Cartesian coordinates. The remaining elements are the ordered curvilinear coordinate variables:

```
(C15) PARABOLIC;
```

(D15)

$$[[\frac{U^2 - V^2}{2}, U\ V], U, V]$$

First we use the function SCALEFACTORS to derive a set of global scale factors. Then we use the function EXPRESS to express its argument in the corresponding coordinate system:

```
(C16) SCALEFACTORS(PARABOLIC) $

(C17) EXAMPLE:  EXPRESS(EXAMPLE);
```

(D17)

$$\frac{\%PI\ (\frac{d^2}{dV^2} (S + H) + \frac{d^2}{dU^2} (S + H))}{V^2 + U^2} =$$

$$\frac{3(\frac{d}{dV} (S\ SQRT(V^2 + U^2)\ P_V) + \frac{d}{dU} (S\ P_U\ SQRT(V^2 + U^2)))}{V^2 + U^2}$$

Alternatively, the global scale factors can be established or changed by supplying the coordinate system as a second argument to EXPRESS rather than an argument to SCALEFACTORS.

Suppose that H depends only on U, that P depends only upon V, and that S depends upon both U and V. To expand the above derivatives, taking advantage of these simplifications:

```
(C18) DEPENDS([S,H],U, [S,P],V) $

(C19) EXAMPLE, DIFF;
```

$$(\mathrm{D19})\quad \frac{\%\mathrm{PI}\left(\frac{d^2 S}{dV^2} + \frac{d^2 S}{dU^2} + \frac{d^2 H}{dU^2}\right)}{V^2 + U^2} = 3\ \left(S\ \mathrm{SQRT}(V^2 + U^2)\ \left(\frac{d}{dV}\ P_V\right)\right.$$

$$+ \frac{dS}{dV}\ \mathrm{SQRT}(V^2 + U^2)\ P_V + \frac{S\ V\ P_V}{\mathrm{SQRT}(V^2 + U^2)} + \frac{dS}{dU}\ P_U\ \mathrm{SQRT}(V^2 + U^2)$$

$$\left. + \frac{S\ U\ P_U}{\mathrm{SQRT}(V^2 + U^2)}\right)/(V^2 + U^2)$$

Now, suppose that we are given the following parabolic-coordinate components of a gradient vector,

```
(C20) EXAMPLE:  [(2*U*V**3+3*U**3*V)/(V**2+U**2),
   (2*U**2*V**2+U**4)/(U**2+V**2)];
```

$$(\mathrm{D20})\qquad \left[\frac{2\ U\ V^3 + 3\ U^3\ V}{V^2 + U^2},\ \frac{2\ U^2\ V^2 + U^4}{V^2 + U^2}\right]$$

and we wish to determine the corresponding scalar potential relative to the potential at the point [0,0]:

```
(C21) POTENTIAL(EXAMPLE);
```

$$(\mathrm{D21})\qquad U^2\ V\ \mathrm{SORT}(V^2 + U^2)$$

There is an analogous function named VECTORPOTENTIAL that computes the vector potential associated with a given curl vector.

## TECHNIQUES

Vector algebra has an intriguing structure. Besides containing the ordinary scalar operations, vector algebra has two special products with somewhat bizarre properties. Although the dot and cross products are both distributive with respect to vector addition, and although scalar factors in either operand may be factored out of the dot and cross product:

1. Vectors are not closed under the dot operation. ($p \cdot q$ is a scalar.)

2. Vectors are closed under the cross operation only in three-dimensional space, the cross product being undefined otherwise.

3. The dot product is commutative

$$p \cdot q \equiv q \cdot p, \tag{1}$$

but the cross product is anticommutative

$$p \times q \equiv -q \times p \tag{2}$$

4. Neither is associative. ($\mathbf{p}\times(\mathbf{q}\times\mathbf{r})\not\equiv(\mathbf{p}\times\mathbf{q})\times\mathbf{r}$, whereas $\mathbf{p}\cdot(\mathbf{q}\cdot\mathbf{r})$ and $(\mathbf{p}\cdot\mathbf{q})\cdot\mathbf{r}$ are invalid.)

5. Neither has a multiplicative unit. (There does not exist a fixed $\mathbf{u}$ such that for arbitrary $\mathbf{p}$, $\mathbf{u}\times\mathbf{p}=\mathbf{p}$ or $\mathbf{p}\times\mathbf{u}=\mathbf{p}$ or $\mathbf{u}\cdot\mathbf{p}=\mathbf{p}$ or $\mathbf{p}\cdot\mathbf{u}=\mathbf{p}$.)

6. Both admit zero divisors. (For all nonzero $\mathbf{p}$,

$$\mathbf{p}\times\mathbf{p} = 0 \ , \tag{3}$$

and there exist nonzero $\mathbf{q}$ such that $\mathbf{p}\cdot\mathbf{q}=0$).

7. Both are connected via ordinary scalar multiplication, denoted with "*", by the strange side relation

$$\mathbf{p}\times(\mathbf{q}\times\mathbf{r}) \equiv (\mathbf{p}\cdot\mathbf{r})*\mathbf{q}-(\mathbf{p}\cdot\mathbf{q})*\mathbf{r} \tag{4}$$

8. The structure is even more complicated if we consider dyadics, triadics, etc.

Vector calculus is equally rich in comparison to its scalar counterpart. Besides containing the usual derivatives, vector calculus has three special differential operators. Although the gradient, divergence, and curl are outative (for example, grad(constant*$\phi$) $\equiv$ constant*(grad $\phi$)) and additive (for example, grad($\phi+\psi$)$\equiv$grad $\phi$ + grad $\psi$):

1. The gradient and divergence are not closed. (The gradient of a scalar is a vector, and the divergence of a vector is scalar.)

2. Vectors are closed under the curl operation only in three-dimensional space, the curl being undefined otherwise.

3. Compositions of these operators do not generally commute, but they do satisfy the following identities

$$\mathrm{curl}(\mathrm{grad}\ \phi) \equiv 0 \ , \tag{5}$$

$$\mathrm{div}(\mathrm{curl}\ \mathbf{p}) \equiv 0 \ , \tag{6}$$

$$\mathrm{curl}(\mathrm{curl}\ \mathbf{p}) \equiv \mathrm{grad}(\mathrm{div}\ \mathbf{p}) + \mathrm{div}(\mathrm{grad}\ \mathbf{p}) \ . \tag{7}$$

Here $\phi$ denotes a scalar, the gradient of a vector is a dyadic, and the divergence of a dyadic is a vector.

4. When applied to various products, most of these operators have expansions similar but not identical to the ordinary derivative of an ordinary product:

$$\mathrm{grad}(\phi\ \mathbf{p}) \equiv \mathbf{p}\ \mathrm{grad}\ \phi + \phi\ \mathrm{div}\ \mathbf{p} \ , \tag{8}$$

$$\mathrm{div}(\phi\ \mathbf{p}) \equiv (\mathrm{grad}\ \phi)\cdot\mathbf{p} + \phi\ \mathrm{div}\ \mathbf{p} \ , \tag{9}$$

$$\mathrm{curl}(\phi\ \mathbf{p}) \equiv (\mathrm{grad}\ \phi)\times\mathbf{p} + \phi\ \mathrm{curl}\ \mathbf{p} \ , \tag{10}$$

$$\mathrm{grad}(\mathbf{p}\times\mathbf{q}) \equiv (\mathrm{grad}\ \mathbf{p})\times\mathbf{q} + (\mathrm{grad}\ \mathbf{q})\dot{\times}\mathbf{p} \ , \tag{11}$$

$$\mathrm{div}(\mathbf{p}\times\mathbf{q}) \equiv \mathbf{q}\cdot(\mathrm{curl}\ \mathbf{p}) + \mathbf{p}\cdot(\mathrm{curl}\ \mathbf{q}) \ , \tag{12}$$

$$\mathrm{curl}(\mathbf{p}\times\mathbf{q}) \equiv \mathbf{q}\cdot(\mathrm{grad}\ \mathbf{p}) - \mathbf{p}\cdot(\mathrm{grad}\ \mathbf{q}) + \mathbf{p}\ \mathrm{div}\ \mathbf{q} - \mathbf{q}\ \mathrm{div}\ \mathbf{p}\ , \tag{13}$$

$$\mathrm{grad}(\mathbf{p}\cdot\mathbf{q}) \equiv (\mathrm{grad}\ \mathbf{p})\cdot\mathbf{q} + (\mathrm{grad}\ \mathbf{q})\cdot\mathbf{p}\ . \tag{14}$$

For brevity, the composition div grad is often denoted as the Laplacian operator:

$$\mathrm{Laplacian}\ \phi \equiv \mathrm{div}\ \mathrm{grad}\ \phi \equiv \nabla^2\phi\ . \tag{15}$$

The Laplacian inherits the linearity of div and grad, together with the following expansion for product operands:

$$\begin{aligned}\mathrm{Laplacian}(\phi\psi) = \phi\ \mathrm{Laplacian}\ \psi &+ 2(\mathrm{Laplacian}\ \psi)(\mathrm{Laplacian}\ \psi)\\ &+ \psi\ \mathrm{Laplacian}\ \phi\ .\end{aligned} \tag{16}$$

For many physical problems, symmetries or boundary surfaces encourage the use of orthogonal curvilinear coordinates that are not rectangular Cartesian. For example, toroidal coordinates are most appropriate for many controlled fusion problems, and oblate spheroidal coordinates are most appropriate for some geophysical problems. In such instances, it is often necessary to know the specific partial differential representation of the gradient, divergence, curl, or Laplacian in order to derive the differential equations pertaining to the desired coordinates.

If the orthogonal curvilinear coordinates are denoted by $\theta_1,\theta_2,\ldots,\theta_n$, and a transformation to some rectangular Cartesian coordinates $x_1,x_2,\ldots,x_m$, with $m\geq n$, is given by

$$x_j = f_j(\theta_1,\theta_2,\ldots,\theta_n),\quad (j=1,2,\ldots,n)\ , \tag{17}$$

then coordinate scale factors are defined by

$$h_k = [\sum_{j=1}^{m} (\frac{\partial f_j}{\partial\theta_k})^2]^{1/2}\ , \qquad (k=1,2,\ldots,n)\ . \tag{18}$$

Moreover

$$\sum_{j=1}^{m} \frac{\partial f_j}{\partial\theta^k}\frac{\partial f_j}{\partial\theta^\ell} \equiv 0\ , \qquad (k\neq\ell)\ . \tag{19}$$

Otherwise the coordinates are nonorthogonal. The function SCALEFACTORS attempts to verify equation (19), printing a warning together with the simplified left-hand side when it does not succeed in doing so. This precaution revealed an error in the alleged-orthogonal confocal-paraboloidal coordinates listed in the reference tables of two widely used vector-analysis texts! Computer algebra is advisable for checking even when a published "answer" is available.

Most of the classic orthogonal coordinate-transformation examples of equation (17), involve trigonometric functions and/or hyperbolic functions and/or square roots. Thus, there is a vital need for effective trigonometric, hyperbolic, and fractional-power simplification during the evaluation of formulas (18) and (19). The built-in RADCAN function provided the fractional-power simplification, but it was necessary to develop a new trigonometric/hyperbolic simplifier, different from those built-in. Though crucial to the performance of this portion of the vector package, a sufficiently thorough discussion of this new simplifier would lead us too far astray here, so the simplifier is discussed separately in reference 1.

Let

$$H = \prod_{k=1}^{n} h_k \; . \qquad (20)$$

Then, using ordered lists to represent the components of a vector, the general formulas for the gradient, divergence, and Laplacian are

$$\text{grad } \phi = [\frac{1}{h_1}\frac{\partial\phi}{\partial\theta_1}, \frac{1}{h_2}\frac{\partial\phi}{\partial\theta_2}, \ldots, \frac{1}{h_n}\frac{\partial\phi}{\partial\theta_n}] \; , \qquad (21)$$

$$\text{div } \mathbf{p} = \frac{1}{H}\sum_{k=1}^{n}\frac{\partial}{\partial\theta_k}\left(\frac{Hp_k}{h_k}\right) \; , \qquad (22)$$

$$\text{Laplacian } \mathbf{p} = \frac{1}{H}\sum_{k=1}^{n}\frac{\partial}{\partial\theta_k}\left(\frac{Hp_k}{h_k^2}\right) \; . \qquad (23)$$

For $n=3$, the general formula for the curl is

$$\text{curl } \mathbf{p} = [\frac{h_1}{H}(\frac{\partial}{\partial\theta_2}(h_3p_3) - \frac{\partial}{\partial\theta_3}(h_2p_2)),$$

$$\frac{h_2}{H}(\frac{\partial}{\partial\theta_3}(h_1p_1) - \frac{\partial}{\partial\theta_1}(h_3p_3)), \; \frac{h_3}{H}(\frac{\partial}{\partial\theta_1}(h_2p_2) - \frac{\partial}{\partial\theta_2}(h_1p_1))]. \qquad (24)$$

The symbolic differentiation and algebra necessary for evaluating formulas (18) through (24) is straightforward but tedious -- an ideal computer-algebra application. In fact, after completing this vector-analysis package I discovered that a package similar to this curvilinear-components portion had already been written by Martin S. Cole.

It is sometimes desired to compute the inverse of these differential operations: Given a specific vector field, find a field, if one exists, for which the given field is the gradient or curl.

If a given vector $\mathbf{p}$ is the gradient of an unknown scalar potential $\phi$, then denoting an arbitrary point by $\underline{\hat{\theta}}=(\hat{\theta}_1,\hat{\theta}_2,\ldots,\hat{\theta}_n)$,

$$\phi(\underline{\theta}) = \phi(\hat{\underline{\theta}}) + h_1\int_{\hat{\theta}_1}^{\theta_1} p_1(\theta_1,\theta_2,\ldots,\theta_n)d\theta_1 + h_2\int_{\hat{\theta}_2}^{\theta_2} p_2(\hat{\theta}_1,\theta_2,\theta_3,\ldots,\theta_n)d\theta_2$$

$$+ \ldots + h_n\int_{\hat{\theta}_n}^{\theta_n} p_n(\hat{\theta}_1,\hat{\theta}_2,\ldots,\hat{\theta}_{n-1},\theta_n)d\theta_n , \qquad (25)$$

where $\phi(\hat{\underline{\theta}})$ is an undeterminable constant.

Successful closed-form computation of these integrals may depend upon the chosen $\hat{\underline{\theta}}$ and the chosen ordering of the components of $\underline{\theta}$. The validity of this formula depends upon the assumed existence of a scalar potential. Consequently, the function POTENTIAL attempts to use differentiation and simplification to verify any candidate constructed by this formula.

If a given vector **p** is the curl of an unknown three-dimensional vector potential **q**, then

$$\mathbf{q} = [\frac{1}{h_1}\int_{\hat{\theta}_3}^{\theta_3} h_1 h_3 p_2(\theta_1,\theta_2,\theta_3)d\theta_3 - \frac{1}{h_1}\int_{\hat{\theta}_2}^{\theta_2} h_1 h_2 p_3(\theta_1,\theta_2,\hat{\theta}_3)d\theta_2 ,$$

$$- \frac{1}{h_2}\int_{\hat{\theta}_3}^{\theta_3} h_2 h_3 p_1(\theta_1,\theta_2,\theta_3)d\theta_3 , 0] + \text{grad } \psi , \qquad (26)$$

where $\psi$ is an arbitrary twice-differentiable scalar potential. Successful closed-form computation of these integrals may depend upon the chosen $\hat{\theta}_2$ and $\hat{\theta}_3$ together with a well-chosen cyclic permutation of the components of $\underline{\theta}$. The validity of this formula depends upon the assumed existence of a vector potential. Consequently, the function VECTORPOTENTIAL attempts to use differentiation and simplification to verify any candidate constructed by this formula. Formulas (25) and (26) are generalizations of those given in pages 201-202 of reference 2. For the program, $\hat{\underline{\theta}}$ in equations (25) and (26) is specified by the global variable POTENTIALZEROLOC, which is initially set to [0,0,...,0].

MACSYMA has several built-in features which greatly facilitate the implementation of extensions such as this vector package:

1. The syntax extension facility makes it easy to introduce new operators, such as "×", GRAD, DIV, CURL, and LAPLACIAN, together with their parse binding powers and restrictions on their valid operand types. However, attempted implementation of GRAD, DIV, CURL, and LAPLACIAN respectively as DEL, DEL ·, DEL × and DEL↑2 caused incredible chaos, which should not be surprising to anyone who has written an extendable parser.

2. The declaration facility made it easy to establish the automatic outative and optional additive properties of GRAD, DIV, CURL, and LAPLACIAN. The declaration facility also made it easy to supplement the algebraic properties of the built-in operator "•" with commutativity.

3. A built-in flag permitted defeat of the default associativity property of "•", and another built-in flag provided optional distribution of "•" over "+". A built-in flag also permitted the automatic factoring of scalars from dot operands.

4. The pattern-matching automatic-substitution facility made it easy to implement simplifications such as transformations (3), (5) and (6).

5. The procedure-definition facility together with a built-in function for determining the parts of expressions made it possible to implement the other expansions and simplifications without recourse to the lower-level MACSYMA implementation language.

Simplifications that are unlikely to enlarge an expression, that do not drastically change the form of an expression, and that are easy to implement via declaration and automatic pattern-matching substitutions were made automatic. Examples include the use of transformations (1), (3), (5) and (6).

Other expansions, such as expansions (2), (4), (7), and (8) through (27), together with the employment of additivity or distributivity require a specific request by the user, via the function VECTORSIMP, together perhaps with the appropriate setting of various global variables.

It is expected that most users will wish to use the function VECTORSIMP with the flag EXPANDALL set to its default value of FALSE, requesting only the least controversial expansions, or set to TRUE, requesting nearly every programmed expansion. However, for the user who needs fine control there is a hierarchy of flags permitting individual control over each of the programmed expansions or over various logical groupings of these. The flags are

- EXPANDALL,
    - EXPANDDOT,
        - EXPANDDOTPLUS,
    - EXPANDCROSS,
        - EXPANDCROSSPLUS,
        - EXPANDCROSSCROSS,
    - EXPANDGRAD,
        - EXPANDGRADPLUS,
        - EXPANDGRADPROD,
    - EXPANDDIV,
        - EXPANDDIVPLUS,
        - EXPANDDIVPROD,

EXPANDCURL,
    EXPANDCURLPLUS,
    EXPANDCURLCURL,
EXPANDLAPLACIAN,
    EXPANDLAPLACIANPLUS,
    EXPANDLAPLACIANPROD.

The PLUS suffix refers to employing additivity or distributivity. The PROD suffix refers to the expansion for an operand that is any kind of product. EXPANDCROSSCROSS refers to expansion (4), and EXPANDCURLCURL refers to expansion (7). EXPANDCROSS=TRUE has the same effect as EXPANDCROSSPLUS= EXPANDCROSSCROSS=TRUE, etc. Two other flags, EXPANDPLUS AND EXPANDPROD, have the same effect as setting all similarly suffixed flags true. When TRUE, another flag named EXPANDLAPLACIANTODIVGRAD, replaces the LAPLACIAN operator with the composition DIV GRAD. For convenience the flags have all been declared EVFLAGS.

Those who prefer a plethora of functions to a plethora of flags are encouraged to define a corresponding set of functions which merely locally set the appropriate flag, then use VECTORSIMP. Those who loathe both approaches are free to ignore all of this.

## TEST RESULTS

A crucial question is: How complicated can problems be, for the various portions of the vector package, before exhausting the available memory space or a reasonable amount of computing time? Unfortunately, the answer to this question is very problem-dependent, difficult to characterize concisely and objectively. However, to suggest rough indications, this section summarizes a variety of test results.

First, to test the non-component simplifications, default simplification, followed by VECTORSIMP with EXPANDALL=TRUE, was applied to the expressions in Table 1, taken from pages 32-33, 60, and 215 of reference 2.

These examples all correctly simplified to zero, with the exception of case 6, which simplified to

$$-a\cdot c\times(a\cdot b\times c\ast b-a\cdot b\times(b\times c))-(a\cdot b\times c)^2$$

A second application successfully annihilated the term containing $b\times(b\times c)$, and rearranged the first term to give

$$a\cdot(a\cdot b\times c\ast b)\times c-(a\cdot b\times c)^2 \ .$$

$a\cdot b\times c$ could be factored out, clearly revealing that the expression is zero, but the built-in scalar-factoring-out mechanism does not recognize that $a\cdot b\times c$ is a scalar despite its vector components.

Regarding the orthogonal curvilinear components portion of the package, Table 2 reports the times required to compute the scale factors, and express three particular expressions in a variety of three-dimensional coordinate systems. The first expression is an equation arising in magnetohydrodynamics given in reference 3:

$$\text{Laplacian Laplacian } \mathbf{w} + \text{Curl}(\eta \text{ Curl } \mathbf{w}) = -\text{Curl } \underline{\xi} \ . \tag{27}$$

The second expression is the Navier-Stokes equation of fluid mechanics:

$$\frac{\partial \mathbf{v}}{\partial t} = \nu \text{ Laplacian } \mathbf{v} - \mathbf{v}\cdot\text{grad } \mathbf{v} + \frac{\nu}{3} \text{ grad div } \mathbf{v} - \frac{\text{grad } p}{\rho} \ . \tag{28}$$

The third expression is all but one term of another equation from magnetohydrodynamics, given in reference 4:

$$\frac{\partial \mathbf{B}}{\partial t} = \text{curl}(\mathbf{v}\times\mathbf{B}) - \frac{c}{4\pi e} \text{ curl } \left(\frac{(\text{curl } \mathbf{B})\times\mathbf{B}}{N_e}\right) - \frac{ck}{eN_e} (\text{grad } N_e)\times(\text{grad } T_e) \ . \tag{29}$$

The omitted term was

$$\text{curl}\left(\frac{c^2}{4\pi} \mathbf{r}\cdot(\text{curl } \mathbf{B})\right) , \tag{30}$$

where $\mathbf{r}$ is a resistivity dyadic. Although the vector package fortuitously represents the gradient of a vector as a list of derivatives of lists, which can be interpreted as a dyadic, the package was not designed to treat dyadics in general. The function EXPRESS expands expressions into components from the inside out, and expansion of the curl operator requires an argument that is a list of three elements. Thus, EXPRESS halts with an error message when it tries to expand the outer curl in expression (30).

The definitions of the coordinate systems are given in reference 5. As indicated in Table 2, the scale-factor computation depends strongly on the complexity of the coordinate system, whereas the time required to express vector expressions does not.

To test the function named POTENTIAL, the fully-expanded gradient of each of the expressions in Table 3 was derived in three-dimensional rectangular Cartesian coordinates. Then, with POTENTIALZEROLOC set as indicated, POTENTIAL was applied in an attempt to generate an expression differing from the original by no more than a constant.

For case 2, POTENTIAL printed a warning that it could not verify the solution by differentiation together with simplification. However, expansion of the trigonometric factor of the gradient revealed the answer was correct. (The integration utilized this expansion, whereas the verification simplification did not.)

# AUTHOR INDEX

Abdali . . . 253 3/F10
Ament . . . 87 2/A7
Andersen . . . 161 2/F11
Anderson . . . 395 5/C8
Avgoustis . . . 21 1/C6
Bogen . . . 11, 75 1/B10
Bulnes . . . 461 6/A9
Caviness . . . 253 3/F10
Char . . . 53 1/E10
Cohen . . . 275 4/A9
Cuthill . . . 131 2/E3
Doohovskoy . . . 473 6/B7
Fateman . . . 43, 327 1/D14
Fennelly . . . 97 2/B3
Geddes . . . 405 5/D4
Genesereth . . . 291, 309 4/B11
Golden . . . 1, 109 1/A14
Gosper . . . 237 3/E8
Grünbaum . . . 491 6/C11
Gupta . . . 151 2/F1
Ivie . . . 317 4/D9
Karney . . . 377 5/B4
Kulp . . . 385 5/B12
Lafferty . . . 347 4/F11
Lau . . . 395 5/C8
Lewis . . . 277 4/A11
Moenck . . . 225 3/D10
Moses . . . 123, 275 2/D1
Ng . . . 151, 177 2/F1
Noor . . . 161 2/F11
Pavelle . . . 75, 97 1/G4
Poole . . . 145 2/E9
Pridor . . . 253 3/F10
Rothstein . . . 263 3/G6
Spear . . . 369 5/A10
Steele . . . 203, 215 3/C2
Stoutemyer . . . 315, 425, 447 4/D7
Wahlquist . . . 71 1/F14
Wang . . . 55, 435 1/E12
Yun . . . 65 1/F8
Zippel . . . 361 4/G11

# CONTENTS

FOREWORD . . . . . . . . . . . . . . . . . . . . . . . . . . . . . iii 1/A10

PREFACE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . V 1/A12

AUTHOR INDEX . . . . . . . . . . . . . . . . . . . . . . . . . . . vii 1/A1

1. MACSYMA'S SYMBOLIC ORDINARY DIFFERENTIAL EQUATION SOLVER . . . . . . . 1 1/A14
Jeffrey P. Golden

2. A PROGRAM FOR THE SOLUTION OF INTEGRAL EQUATIONS . . . . . . . . . . . 11 1/B10
Richard A. Bogen

3. SYMBOLIC LAPLACE TRANSFORMS OF SPECIAL FUNCTIONS . . . . . . . . . . . 21 1/C6
Yannis Avgoustis

4. AN IMPROVED ALGORITHM FOR THE ISOLATION OF POLYNOMIAL REAL ZEROS . . . 43 1/D14
Richard J. Fateman

5. FLOATING POINT ISOLATION OF ZEROS OF A REAL POLYNOMIAL VIA MACSYMA . . 53 1/E10
Bruce W. Char

6. PRESERVING SPARSENESS IN MULTIVARIATE POLYNOMIAL FACTORIZATION . . . . 55 1/E12
Paul S. Wang

7. ON THE EQUIVALENCE OF POLYNOMIAL GCD AND SQUAREFREE FACTORIZATION PROBLEMS . . . . . . . . . . . . . . . . . . . . . . . . . . 65 1/F8
David Y. Y. Yun

8. DIFFERENTIAL FORM ANALYSIS USING MACSYMA . . . . . . . . . . . . . . . 71 1/F14
Hugo D. Wahlquist

9. INDICIAL TENSOR MANIPULATION ON MACSYMA . . . . . . . . . . . . . . . . 75 1/G4
Richard A. Bogen and Richard Pavelle

10. PURE FIELD THEORIES AND MACSYMA ALGORITHMS . . . . . . . . . . . . . . 87 2/A7
William S. Ament

11. BLACK HOLES AND RELATIVISTIC GRAVITY THEORIES . . . . . . . . . . . . . 97 2/B3
A. J. Fennelly and Richard Pavelle

12. THE EVALUATION OF ATOMIC VARIABLES IN MACSYMA . . . . . . . . . . . . . 109 2/C1
Jeffrey P. Golden

13. THE VARIETY OF VARIABLES IN MATHEMATICAL EXPRESSIONS . . . . . . . . 123 2/D1
Joel Moses

14. RATIONAL APPROXIMATION TO $e^{-x}$ WITH NEGATIVE REAL POLES . . . . . . . 131 2/D9
Elizabeth Cuthill

15. TIMING FORMULAS FOR DISSECTION ALGORITHMS ON VECTOR COMPUTERS . . . . 145 2/E9
W. G. Poole, Jr.

16. SYMBOLIC CALCULATIONS IN A FINITE DYNAMIC ELEMENT ANALYSIS . . . . . 151 2/F1
Kajal K. Gupta and Edward W. Ng

17. SYMBOLIC MANIPULATION TECHNIQUES FOR VIBRATION ANALYSIS OF LAMINATED ELLIPTIC PLATES . . . . . . . . . . . . . . . . . . . . 161 2/F11
C. M. Andersen and Ahmed K. Noor

18. OBSERVATIONS ON APPROXIMATE INTEGRATIONS . . . . . . . . . . . . . 177 2/G13
Edward W. Ng

19. LISP: PROGRAM IS DATA — A HISTORICAL PERSPECTIVE ON MACLISP . . . . 181 3/A8
Jon L White

20. LISP: DATA IS PROGRAM — A TUTORIAL IN LISP . . . . . . . . . . . . 191 3/B4
Jon L White

21. DATA REPRESENTATIONS IN PDP-10 MACLISP . . . . . . . . . . . . . . 203 3/C2
Guy Lewis Steele Jr.

22. FAST ARITHMETIC IN MACLISP . . . . . . . . . . . . . . . . . . . . 215 3/C14
Guy Lewis Steele Jr.

23. ON COMPUTING CLOSED FORMS FOR SUMMATIONS . . . . . . . . . . . . . 225 3/D10
Robert Moenck

24. INDEFINITE HYPERGEOMETRIC SUMS IN MACSYMA . . . . . . . . . . . . 237 3/E8
R. Wm. Gosper, Jr.

25. MODULAR POLYNOMIAL ARITHMETIC IN PARTIAL FRACTION DECOMPOSITION . . . 253 3/F10
S. K. Abdali, B. F. Caviness, and A. Pridor

26. A NEW ALGORITHM FOR THE INTEGRATION OF EXPONENTIAL AND LOGARITHMIC FUNCTIONS . . . . . . . . . . . . . . . . . . . . . . . 263 3/G6
Michael Rothstein

27. SUMMATION OF RATIONAL EXPONENTIAL EXPRESSIONS IN CLOSED FORM . . . . 275 4/A9
Joel Moses and Jacques Cohen

28. USER AIDS FOR MACSYMA . . . . . . . . . . . . . . . . . . . . . . . . . 277 4/A11
V. Ellen Lewis

29. THE DIFFICULTIES OF USING MACSYMA AND THE FUNCTION OF USER AIDS . . . 291 4/B11
Michael R. Genesereth

30. AN AUTOMATED CONSULTANT FOR MACSYMA . . . . . . . . . . . . . . . . . . 309 4/D1
Michael R. Genesereth

31. A MACSYMA COMPUTER-ALGEBRA MOVIE DEMONSTRATION . . . . . . . . . . . 315 4/D7
David R. Stoutemyer

32. SOME MACSYMA PROGRAMS FOR SOLVING DIFFERENCE EQUATIONS . . . . . . . 317 4/D9
John Ivie

33. SOME COMMENTS ON SERIES SOLUTIONS . . . . . . . . . . . . . . . . . . . 327 4/E5
Richard J. Fateman

34. POWER SERIES SOLUTIONS OF ORDINARY DIFFERENTIAL EQUATIONS IN MACSYMA . . . . . . . . . . . . . . . . . . . . . . . . . . . 347 4/F11
Edward L. Lafferty

35. RADICAL SIMPLIFICATION MADE EASY . . . . . . . . . . . . . . . . . . . 361 4/G11
Richard E. B. Zippel

36. A CONSTRUCTIVE APPROACH TO COMMUTATIVE RING THEORY . . . . . . . . . 369 5/A10
David A. Spear

37. REDUCTION OF THE EQUATION FOR LOWER HYBRID WAVES IN A PLASMA TO A NONLINEAR SCHRÖDINGER EQUATION . . . . . . . . . . . . . . . . . 377 5/B4
Charles F. F. Karney

38. RAY TRAJECTORIES IN A TORUS: AN APPLICATION OF MACSYMA TO A COMPLEX NUMERICAL COMPUTATION . . . . . . . . . . . . . . . . . . . . . . . . 385 5/B12
John L. Kulp

39. APPLICATION OF MACSYMA TO FIRST ORDER PERTURBATION THEORY IN CELESTIAL MECHANICS . . . . . . . . . . . . . . . . . . . . . . . . 395 5/C8
John D. Anderson and Eunice L. Lau

40. SYMBOLIC COMPUTATION OF RECURRENCE EQUATIONS FOR THE CHEBYSHEV SERIES SOLUTION OF LINEAR ODE's . . . . . . . . . . . . . . . . . . . 405 5/D4
K. O. Geddes

41. sin(x)**2 + cos(x)**2 = 1 . . . . . . . . . . . . . . . . . . . . . . . 425 5/E10
David R. Stoutemyer

42. MATRIX COMPUTATIONS IN MACSYMA . . . . . . . . . . . . . . . . . . . . 435 5/F6
Paul S. Wang

43. SYMBOLIC COMPUTER VECTOR ANALYSIS . . . . . . . . . . . . . . . . . . . 447 5/G4
David R. Stoutemyer

44. A NATURAL WAY TO DO SPATIAL LINEAR GEOMETRY IN MACSYMA . . . . . . . 461 6/A9
Juan Bulnes

45. VARIETIES OF OPERATOR MANIPULATION . . . . . . . . . . . . . . . . . . 473 6/B7
Alexander Doohovskoy

46. PROGRESS REPORT ON THE DETERMINANT OF A RANDOM MATRIX . . . . . . . . 491 6/C11
F. A. Grünbaum

BIBLIOGRAPHY . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 493 6/C13

In contrast, POTENTIAL was able to verify the solution for the similar case 3, which has no angle sum.

## CONCLUSIONS

The examples here demonstrate that vector analysis is a feasible and worthwhile supplementary program package for a computer-algebra system.

## ACKNOWLEDGMENTS

Richard Bogen has been an invaluable critic and teacher, who greatly helped bring this effort to a successful conclusion. Jeffrey Golden is, as always, an unfailing source of good suggestions.

## REFERENCES

1. Stoutemyer, David R.: sin($x$)**2 + cos($x$)**2 = 1, Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper no. 41 of this compilation.)

2. Brand, L.: Vector and Tensor Analysis, John Wiley & Sons, Inc., 1947.

3. Nielson, C.W., and Lewis, R.: Particle-Code Models in the Nonradiative Limit, Methods in Computational Physics, Vol. 16, J. Killeen, ed. Academic Press, Inc., 1976, p. 116.

4. Boris, J.P., and Book, D.L.: Solution of Continuity Equations by the Method of Flux-Corrected Transport, Methods in Computational Physics, Vol. 16, J. Killeen, ed., Academic Press, Inc., 1976, p. 379.

5. Spiegel, M.R.: Vector Analysis, McGraw-Hill Book Co., 1967.

TABLE 1

| Case | Input Expression | Computing Time (seconds) | |
|---|---|---|---|
| | | Default | VECTORSIMP |
| 1 | $(d-a)\cdot(b-c)+(d-b)\cdot(c-a)+(d-c)\cdot(a-b)$ | 0.02 | 0.3 |
| 2 | $(b-a)\cdot(b-a)+(c+b)\cdot(c-b)+(d-c)\cdot(d-c)$<br>$+(a-d)\cdot(a-d)-(c-a)\cdot(c-a)+(d-b)\cdot(d-b)$<br>$+(a+c-b-d)\cdot(a+c-b-d)$ | 0.04 | 0.6 |
| 3 | $(a-b)\cdot(k-\frac{a+b}{2})+(b-c)\cdot(k-\frac{b+c}{2})$<br>$+(c-a)\cdot(k-\frac{c+a}{2})$ | 0.03 | 0.3 |
| 4 | $(a+b-c-d)\cdot(a+b-c-d)$ -<br>$(a-b-c+d)\cdot(a-b-c+d)-4(a-c)\cdot(b-d)$ | 0.02 | 0.7 |
| 5 | $(b\times c)\times(a\times d)+(c\times a)\times(b\times d)+(a\times b)\times(c\times d)$<br>$+2(a\cdot b\times c)+d$ | 1.3 | 4.8 |
| 6 | $(a\times b)\times(b\times c)\cdot(c\times a)-(a\cdot(b\times c))^2$ | 4.0 | 2.0 |
| 7 | $(a-d)\times(b-c)+(b-d)\times(c-a)+(c-d)\times(a-b)$<br>$-2*(a\times b+b\times c+c\times a)$ | 0.5 | 8.6 |

TABLE 2

| Coordinates | time in seconds | | | |
|---|---|---|---|---|
| | Scale Factors | Eq. (27) | Eq. (28) | Eq. (29) |
| parabolic cylindrical | 0.7 | 0.4 | 0.9 | 0.9 |
| rectangular Cartesian | 0.8 | 0.8 | 0.8 | 1.2 |
| polar cylindrical | 1.5 | 0.3 | 0.4 | 0.8 |
| paraboloidal | 3.4 | 0.4 | 0.9 | 1.3 |
| conical | 6.9 | 1.5 | 1.3 | 2.1 |
| spherical | 7.4 | 0.4 | 0.8 | 0.8 |
| elliptic cylindrical | 9.3 | 0.4 | 0.5 | 1.4 |
| confocal ellipsoidal | 17.8 | 1.9 | 1.1 | 2.5 |
| bipolar cylindrical | 20.5 | 0.4 | 0.5 | 1.4 |
| oblate spheroidal | 21.8 | 0.5 | 1.2 | 1.0 |
| prolate spheroidal | 35.3 | 0.5 | 0.5 | 1.5 |
| toroidal | 57.0 | 0.5 | 0.5 | 1.6 |

TABLE 3

| Case | Expression | POTENTIALZEROLOC | time in seconds |
|---|---|---|---|
| 1 | $xy^2z + (x^3+3x)\ y$ | $[x=0,\ y=0,\ z=0]$ | 1.1 |
| 2 | $x^3\sin(ax+b)e^{3y+\pi}\ z^2\log\ (1+z)$ | $[x=0,\ y=0,\ z=0]$ | 11.2 |
| 3 | $x^3\sin(ax)e^{3y+\pi}z^2\ \log\ (1+z)$ | $[x=0,\ y=0,\ z=0]$ | 4.2 |
| 4 | $x/(y+z+1)$ | $[x=0,\ y=0,\ z=0]$ | 1.0 |
| 5 | $(x^2+y^2+z^2)^{-1/2} - 3^{-1/2}$ | $[x=1,\ y=1,\ z=1]$ | 5.9 |
| 6 | $\log_e(x^2+y^2)$ | $[x=1,\ y=0,\ z=0]$ | 1.1 |

# A NATURAL WAY TO DO SPATIAL LINEAR GEOMETRY IN MACSYMA

Juan Bulnes
Stanford Artificial Intelligence Laboratory

## ABSTRACT

A set of routines appropriate for use as an interactive aid in 3-dimensional calculations with planes, lines and points is presented. The mathematical language used is vector calculus. The simplicity with which these routines can be written in MACSYMA is quite remarkable, and that is the main reason for presenting them here. Because of the natural way in which geometric intuition is mapped into them, they can serve as a model for an interactive computational aid for architects.

## INTRODUCTION

This paper is concerned with the application of MACSYMA to 3-dimensional linear geometry calculations. A number of routines are presented which provide a designer with a most natural language for interacting with the system. For example, the designer may be an architect who has drawn tentative plans for a structure which he wishes to meet certain specifications regarding shape, perspectives, etc.; his design having been driven by the outward shape he has in mind, he may know the exact dimensions of some of the subsystems of his structure, but there may be many essential gaps in his knowledge of how they fit together; also he may still be wondering as to which of his givens can be used as initial reference and whether the rest would then be under-, over- or uniquely determined by these. Our routines permit him to interactively explore the consequences of his decisions. In the situation envisioned, the structure does not have any curved surfaces, although it is possible to deal with them, with some extra work.

The mathematical language chosen is three dimensional vector calculus and all surfaces are represented parametrically. Thus a line is represented by a vector depending on one parameter and a plane by one that depends on two parameters. This is different from the usual representation in analytic geometry, where a plane is represented by an equation in three variables and a line by a system of two equations, and where the variables X, Y and Z stand for the three coordinates of a point. In our representation, a point is represented by an ordered triple [a,b,c] and our parameters do not represent coordinates. Thus the vector [X,Y,Z] with three free parameters represents the entire 3-dimensional space, while [0,X,0] represents the y-axis, the same as [0,Y,0] or [0,U,0].

The objects we are dealing with are points, lines and planes. It seems handier to represent an object like a line by a vector with one free parameter rather than by a system of two equations. It will be shown that this representation makes the routines that compute distances and angles extremely simple; in fact they are written in just the language of vector calculus.

The most important convention we have kept throughout is that for any line the free parameter will be named X and for any plane the parameters will be Y and Z. Thus, computing the intersection between two planes can be done by renaming the parameters of one of them to X and U, and then solving the resulting system of three equations for Y,Z and U; the solution will contain X and will therefore be a line.

While the above is a convention for the system we are building in MACSYMA, the following are conventions for the sake of this exposition only. We shall use lower case letters a,b,c,... to represent numerical quantities, as opposed to parameters (however, responses typed back by MACSYMA will appear always in upper case). Thus we may talk about the point [a,b,c], for instance; or about some horizontal plane [Y,Z,a]. Upper case identifiers A,B,..., L1,L2,... and PL1,PL2,... will be used as arguments in the definitions of MACSYMA functions. But X,Y,Z,U will be reserved for the parameter names.

The author wishes to acknowledge his debt to Bill Gosper who taught him how to use MACSYMA and substantially contributed to the system shown in the sequel.

## BASIC VECTOR CALCULUS IN MACSYMA

Vector addition, substraction, multiplication and division by scalars are already built in MACSYMA. So is also the *dot* product. For example:

```
(C1) [a1,a2,a3]+[b1,b2,b3];
(D1)                [B1 + A1, B2 + A2, B3 + A3]

(C2) a*[b1,b2,b3];
(D2)                     [A B1, A B2, A B3]

(C3) [a1,a2,a3].[b1,b2,b3];
(D3)                   A3 B3 + A2 B2 + A1 B1
```

Thus the only basic operation that needs be added is the *cross* product, also called vector product. The following routine suggested by Bill Gosper combines the MACSYMA functions DETERMINANT and MATRIX so as to write the cross product in the very same way it is defined in textbooks.

```
CROSS(A, B) := DETERMINANT(MATRIX([[1, 0, 0], [0, 1, 0], [0, 0, 1]], A, B))
```

Thus:

```
(C4) CROSS([a1,a2,a3],[b1,b2,b3]);
(D4)        [A2 B3 - A3 B2, A3 B1 - A1 B3, A1 B2 - A2 B1]
```

Of course D4 would make a more efficient definition of the cross product. But Gosper's routine is worthy of presentation for its elegance, because it illustrates the capabilities of the MACSYMA language and also for its additional merit that it follows the mnemotechnic rule by which the definition is commonly remembered.

# FUNCTIONS FOR 3-DIMENSIONAL LINEAR GEOMETRY

Using SOLVE in addition to the basic set of operations just described, one can program a set of useful routines for using MACSYMA as an interactive calculational aid, in a language that follows almost verbatim a tutorial exposition of vector calculus. We start with the norm of a vector:

```
NORM(A) := SQRT(A . A)
```

The distance between two points is the norm of the difference vector:

```
DISTANCE(A, B) := NORM(A - B)
```

Vectors of length one are useful for many purposes, for instance for determining angles. The following function,

```
                 A
UNITL(A) :=  -----------
               NORM(A)
```

yields a vector of length one pointing in the same direction as A. While writing such a function is barely justified from the point of view of economy in typing and not at all justified from the point of view of computational efficiency, it seems worthwhile to keep the user in touch with the intuition behind what he is doing.

Passing a line through two points:

```
LINE(A, B) := A + X*(B - A)
```

And a plane through three points:

```
PLANE(A, B, C) := A + Y*(B - A) + Z*(C - A)
```

Getting the point of intersection of a line and a plane:

```
INTERSECTION(L, PL) := EV(L, SOLVE(L - PL, [X, Y, Z]))
```

There are several ways to compute the intersection line between two planes. One possibility is the following routine, suggested by Bill Gosper.

```
PLANEINTERSECTION(PL1, PL2) :=
        BLOCK([INT], INT : SOLVE(PL1- EV(PL2, Y = X, Z = U), [Y, Z, U]), EV(PL1, INT))
```

This function works fine in most cases; but when the planes are parallel, SOLVE fails and gives the message "inconsistent equations", and that is what it should do. The same happens to INTERSECTION when the line and the plane don't intersect. However, PLANEINTERSECTION fails for the following pair of perpendicular planes because of the asymmetry stemming from the fact that we solved for three arbitrary parameters out of the four.

```
(C5) PLANEINTERSECTION([Y,0,Z],[2,Y,Z]);
INCONSISTENT EQUATIONS:(2)
```

Switching around Y and Z in the first argument does not do any good, but, curiously enough, doing it with the second one does:

```
(C6) PLANEINTERSECTION([Y,0,Z],[2,Z,Y]);
SOLUTION
(E6)                    U = 0
(E7)                    Y = 2
(E8)                    Z = X
(D8)                  [2, 0, X]
```

By tracing SOLVE we find the solution to the puzzle:

```
(C9) PLANEINTERSECTION([Y,0,Z],[2,Y,Z]);
1 ENTER SOLVE [[Y - 2, - X, Z - U], [Y, Z, U]]
INCONSISTENT EQUATIONS:(2)
```

What has happened is that the second equation says X = 0, but X is considered a coefficient because it is being solved for [Y,Z,U]. Switching the second argument helps because we then have U = 0, which is O.K. for a variable U.

Failure of PLANEINTERSECTION due to the above situation is a rare occurrence; a more serious problem of this and other routines is occasional numerical unstability. In the next section we shall discuss some modifications that help with the latter; also we will show how to construct a routine for intersecting planes that never fails unless the planes do not intersect.

In the rest of this section we shall use a function VCOEFF instead of the MACSYMA function COEFF. The definition of VCOEFF will be given in the next section, as we see why COEFF does not always work.

The following function GRADVECT computes a vector of unitary length perpendicular to a plane.

```
GRADVECT(PL) := UNITL(CROSS(VCOEFF(PL, Y), VCOEFF(PL, Z)))
```

Similarly, the unitary vector pointing in the direction of a line.

```
UNITDIR(LINE) := UNITL(VCOEFF(LINE, X))
```

The angle between two lines can be computed with help of UNITDIR. The simplest way is the following.

```
ACOS( UNITDIR(L1) . UNITDIR(L2) )
```

However, as the referee suggested, it is numerically preferable to use ATAN2 instead of ACOS or ASIN, as follows.

```
ANGLEBETWEENLINES(L1,L2):=
                    BLOCK([INT1,INT2],INT1:UNITDIR(L1),INT2:UNITDIR(L2),
                          ATAN2(NORM(CROSS(INT1,INT2)),INT1.INT2))
```

This routine computes the correct angle modulo 2π. In practical applications you would probably prefer to compute angles modulo π, because repeated use of the cross product makes it difficult to keep track of the orientation of the different unitary vectors. Also angles are computed in radians, but converting them to degrees is trivial.

A function for computing the shortest distance between two lines is

```
DISTANCEBETWEENLINES(L1, L2) :=
     ABS((EV(L1, X = 0) - EV(L2, X = 0)) . UNITL(CROSS(VCOEFF(L1, X), VCOEFF(L2, X))))
```

which takes the vector from a random point on one line to a random point on the other one and projects it onto the vector perpendicular to both lines. However it fails when the lines are parallel, in which case the appropriate procedure is to take a random point on the first line by an EV(L,X=0), and compute its distance to the other line using the following function.

```
DISTANCEFROMPOINTTOLINE(A, L) :=
                    NORM(CROSS(A - EV(L, X = 0), UNITL(VCOEFF(L, X))))
```

Other useful functions are:

```
DISTANCEFROMPOINTTOPLANE(A, PL) :=
                    ABS((A - EV(PL, Y = 0, Z = 0)) . GRADVECT(PL))

  ANGLELINEWITHPLANE(L, PL) := ABS(π/2 - ACOS(UNITDIR(L) . GRADVECT(PL)))
```

The names of these routines are self explanatory. The following one computes the angle between two planes.

```
SOLIDANGLE(PL1, PL2) := π - ACOS(GRADVECT(PL1) . GRADVECT(PL2))
```

An interesting problem is passing through a point P a plane perpendicular to a line L. It can be solved in the following way: let the vector [X,Y,Z] represent a random point in 3-space; then [X,Y,Z]-P is a vector from P to a random point; restricting [X,Y,Z]-P to being perpendicular to L, we obtain an equation in X,Y,Z; solve it for X and substitute the solution into [X,Y,Z]; the resulting vector depends on Y and Z and represents the plane sought. The following routine embodies this procedure.

```
NORMALPLANE(P, L) := EV([X,Y,Z], SOLVE( ([X, Y, Z] - P) . UNITDIR(L), X ))
```

But this function, like PLANEINTERSECTION, may fail in some cases; i.e., if the first coordinate of P is 0, it will return [X,Y,Z]. Fortunately the following simple modification makes it reliable.

```
NORMALPLANE(P, L) := EV([X+Y-Z, X-Y+Z, -X+Y+Z],
                    SOLVE( ([X+Y-Z, X-Y+Z, -X+Y+Z] - P) . UNITDIR(L), X ))
```

Similarly, given a line L and a point P not on L, we can draw through P a line perpendicular to L and intersecting L in the following way.

```
DRAWPERPLINE(P, L) := LINE(P, EV(L, SOLVE(LINE(P, L) . UNITDIR(L))))
```

However, if we now want to find the point of intersection of L with the perpendicular drawn by DRAWPERPLINE, we often find that they do not intersect. This is due to the errors of numerical approximation: the two lines may miss each other by less than a millionth of an inch. The second argument to LINE in the function definition of DRAWPERPLINE is supposed to determine on L the nearest point to P; I have found that the following way of using differentiation to find the closest point makes the function more friendly.

```
DRAWPERPLINE(P, L) := LINE(P, EV(L, SOLVE(DIFF( (P - L) . (P - L), X, 1))))
```

In a similar way we might continue defining functions for solving many kinds of geometric problems. But we shall leave our account here and discuss some practical issues in the next two sections.

## SOME HINTS ON MAKING THE SYSTEM MORE FRIENDLY

The foregoing routines suffice for most practical calculations. However, you may often want to look at the numerical values of your points or lines. The following value serves to illustrate a problem associated numerical evaluation.

```
(D10) ( 706351256145697026997480181112244808857010 * X
                              + 166652086816795180962878228055354176601317 5 )
                                                       / 1875956846319908260995774089014019351503416
```

```
(C11) %,numer;
(D11)                 0.0
```

To see what has happened, let us look at its floating point representation.

```
(C12) BFLOAT(D10);
(D12) 5.3306130253567711B-43 (7.06351256145697B40 X + 1.666520868167952B42)
```

The solution to this and other problems is to use EXPAND.

```
(C13) EXPAND(D10);

      1767696304486607082432666926495 X     4994657061429447987693815085325
(D13) ---------------------------------- + ---------------------------------
      4694720871186565638135868151629 2     5622348503627836928927866672104

(C14) %,numer;
(D14)                 0.037652852 X + 0.88835778
```

The MACSYMA function COEFF offers an analogous difficulty, as illustrated by the following case.

```
                              X + 5
(D15)                         -----
                                7

(C16) COEFF(%,X);
(D16)                           0

(C17) COEFF( EXPAND(D15), X);

                                1
(D17)                           -
                                7
```

This is the reason why we had to use a function VCOEFF instead of COEFF in the last section. Our definition of VCOEFF is as follows.

```
VCOEFF(V, X) := MAP(LAMBDA([L], COEFF(EXPAND(L), X)), V)
```

In my own experience, the system is quite friendly if one keeps expressions in expanded form and exercises extreme caution with floating point conversions. In the use of EVAL in the routines, one may include the EXPAND argument throughout. When converting a value using numerical evaluation, it is wise to do it always in two steps: first expand it and then evaluate it. Use of EV(%,EXPAND,NUMER) won't do any good; you have to say:

```
(INT:EXPAND(%), EV(INT,NUMER))
```

As for the particular type of failure of PLANEINTERSECTION showed in the previous section, it occurs so seldom that I have preferred to keep it as it is. However, the following routine will never fail unless we encounter a plane whose two COEFFs are linearly dependent - which could have been created by giving three colinear points to the routine PLANE. Also it will return NIL if the two planes are parallel.

```
PLANEINTERSCT(P1, P2) :=

   BLOCK([INT1,INT2,INT3],  INT1:GRADVECT(P1),
            INT2:ABS(VCOEFF(P2, Y). INT1), INT3:ABS(VCOEFF(P2, Z). INT1),
    IF MAX(INT2, INT3) > 0   THEN
      INTERSECTION( IF INT2 > INT3 THEN EV(P2, Y=X, Z=0)  ELSE EV(P2, Y=0, Z=X), P1)
                                              + X * UNITL(CROSS(GRADVECT(P2), INT1))
      ELSE NIL )
```

This routine works by first locating the coefficient of P2 whose direction meets P1 at a steeper angle and taking a line on P2 in the direction of that coefficient; the point of intersection of this line with P1 is then used as a starting point for the line of intersection of the two planes, which points in the direction of the cross product of the GRADVECTs of the two planes.

# THE USE OF THE SYSTEM: AN EXAMPLE FROM APOLLONIUS.

The referees have expressed the desire to see some examples of the use of the system described in the previous sections. Also one of them raised the question whether there are problems in which the symbolic capability of MACSYMA offers a clear advantage.

To me, the main advantage of the system is its flexibility. If you need to get started on some calculations of your own, here you have an environment where you can compute things exactly as you want. Not having had much experience with other systems for this purpose, I can't give a comparative answer. I hope that the example shown below will permit the experienced user to draw his own conclusions.

As for the question on the symbolic capability, my answer is a qualified yes. I have found examples where it is useful; but in many other cases I have found it necessary to force MACSYMA to stick with numerical, approximated values. Thus I will make a case both ways. I hope that the example worked out as well as the problem of the quarter cylinder mentioned below, will make the reader enthusiastic about symbolic calculation. I can think of examples which make much heavier use of this facility. On the other hand, I hope to temper the enthusiasm so that symbolic computation will not be abused, because the complexity of algebraic expressions grows extremely large in three dimensional calculations and in many cases they will blow up MACSYMA's storage capacity.

For example, consider the following two problems. First give yourself three points P:[p1,p2,0], Q:[q1,q2,0] and R:[r1,r2,0], and compute the coordinates of the center CNT of the circumscribed circle of the triangle. Then let MACSYMA do a RATSIMP on DISTANCE(CNT,P)-DISTANCE(CNT,Q), and it will compute 0. Now give yourself four points with symbolic coordinates in space and compute the coordinates of the center CNT of the circumscribed sphere. You will get a huge expression for each coordinate of CNT. When I asked for RATSIMP(DISTANCE(CNT,Q)-DISTANCE(CNT,P)), MACSYMA was not able to handle it.

When doing practical calculations, it pays to keep values stored in numerical form so as to minimize the size of expressions. Granted this, I have found that a limited use of the symbolic capability can be very useful. For instance, consider the following problem. You want to make a piece in the shape of a quarter of a cylinder that should be inserted between two planes A, B that are not parallel, and the axis of the cylinder is not perpendicular to either of the planes. The planes, the radius and the axis of the cylinder are given; so are also the planes F1, F2 of the two non curved faces of the quarter cylinder. You want to make your cylinder by rolling up a sheet of metal, which should be cut for you on order. Then you may use MACSYMA as follows. Define a line on the cylinder depending on one parameter THETA; THETA is the angle that the plane through LINE(THETA) and through the axis makes with F1. You are interested in the range 0≤THETA≤π/2. Now you can compute the intersections IA(THETA) and IB(THETA), of LINE(THETA) with A and B, respectively. Similarly let IR(THETA) be the intersection of LINE(THETA) with some reference plane perpendicular to the cylinder axis. The distance on the cylinder surface from LINE(THETA) to the edge on F1, is THETA times the radius. With all these functions of THETA, you can now plot the shape of the sheet of metal, which you want cut so that it will fit into your structure. It cannot be overemphasized that for an application of this nature, it is convenient to keep everything but THETA in numerical form.

Now let us look at a sample problem. Presenting any practical application in a short paper like this, I am forced to restrict MACSYMA's output to its shortest possible form. For this reason, I will make use of the following function.

```
NUMVAL(A) := BLOCK([TMP], TMP : EXPAND(A), EV(TMP, NUMER))
```

(I am not claiming there are no better ways of achieving the same effect. Having written this section after my paper was reviewed, I can only apologize if this way of doing it is far from optimal.)

Now consider the following variation of the Apollonius' problems: given two planes PL1 and PL2, and two points A and B, find the center and the radius of a sphere through A and B that is tangent to PL1 and to PL2. We shall take some numerical values for the planes and the points.

```
(C18) PL1 : PLANE([1,0,0], [0,1,0], [0,0,1]);
(D18)                    [- Z - Y + 1, Y, Z]

(C19) PL2 : PLANE([1,0,0], [2,1,0], [2,1,6]);
(D19)                    [Z + Y + 1, Z + Y, 6 Z]

(C20) A : [0,20,20];
(D20)                         [0, 20, 20]

(C21) B : [6,16,16];
(D21)                         [6, 16, 16]
```

Let LOC1 be the locus of the points that are equidistant from A and B. Let LOC2 and LOC3 be the loci of the points that have the same distance to PL1 and to PL2. We use the line of intersection of PL1 and PL2, IL12, as an intermediate value.

```
(C22) LOC1 : NUMVAL( NORMALPLANE( (A+B)/2, LINE(A, B) ));
(D22) [1.33333333 Z - 21, 3.33333334 Z - 2 Y - 21, - 1.33333333 Z + 2 Y + 21]

(C23) IL12 : NUMVAL(PLANEINTERSECTION(PL1, PL2));
(D23)                    [0.75 X + 1, 0.75 X, - 1.5 X]

(C24) LOC2 : NUMVAL( EV(IL12, X=Y) + Z * ( GRADVECT(PL1) + GRADVECT(PL2) ) );
(D24) [1.28445704 Z + 0.75 Y + 1, 0.75 Y - 0.12975651 Z, 0.57735026 Z - 1.5 Y]

(C25) LOC3 : NUMVAL( EV(IL12, X=Y) + Z * ( GRADVECT(PL1) - GRADVECT(PL2) ) );
(D25) [- 0.12975651 Z + 0.75 Y + 1, 1.28445704 Z + 0.75 Y, 0.57735026 Z - 1.5 Y]
```

Intersecting LOC1 with LOC2 and with LOC3, we obtain two lines LOC4 and LOC5, respectively, on which such a sphere may exist. Of course it will exist in at most one of them, but we do not yet know on which one.

```
(C26) LOC4 : NUMVAL(PLANEINTERSECTION(LOC1, LOC2));
(D26) [- 0.87826738 X - 27.657506, 0.914488[illegible] X + 2.8949957, - 2.2318895 X - 12.8812547]
```

```
(C27) LOC5 : NUMVAL(PLANEINTERSECTION(LOC1, LOC3));
(D27) [0.63169204 X - 1.08222031, 1.92112805 X + 20.611853, 9.264817 - 0.97958996 X]
```

Now we proceed to find out whether there is any point on LOC4 or LOC5 that has the same distance to, say, A and PL1.

```
(C28) Q4A : NUMVAL( DISTANCE(LOC4,A)↑2 );
(D28)           6.5889733 X^2  + 164.071367 X + 2138.6957

(C29) Q5A : NUMVAL( DISTANCE(LOC5,A)↑2 );
(D29)           5.0376453 X^2  + 21.8869693 X + 116.789719

(C30) Q41 : NUMVAL( DISTANCEFROMPOINTTOPLANE(LOC4,PL1)↑2 );
(D30)                ABS(1.26766976 X + 22.310988)^2
```

The last line is typical of some of the minor problems one frequently encounters. It is the price one has to pay for using a system of such great generality. It still seems much less than the price one pays with more conventional systems. So we try again.

```
(C31) Q41 : NUMVAL( PART(DISTANCEFROMPOINTTOPLANE(LOC4,PL1),1)↑2 );
(D31)           1.60698665 X^2  + 56.56593 X + 497.78019

(C32) Q51 : NUMVAL( PART(DISTANCEFROMPOINTTOPLANE(LOC5,PL1),1)↑2 );
(D32)           0.8313226 X^2  + 29.262555 X + 257.51048

(C33) REALROOTS(Q41-Q4A);
(D33)                          [ ]

(C34) REALROOTS(Q51-Q5A);
(D35)                      [E34, E35]
```

So we know there is no such sphere on LOC4 but there are two of them on LOC5. Now we proceed to determine their centers and radii.

```
(C36) CNT1 : NUMVAL(EV(LOC5, E34));
(D36)           [- 4.2238372, 11.0574374, 14.1068072]

(C37) CNT2 : NUMVAL(EV(LOC5, E35));
(D37)           [3.1670384, 33.534875, 2.71568334]

(C38) RADIUS1 : NUMVAL(DISTANCE(CNT1, A));
(D38)                     11.5125996

(C39) RADIUS2 : NUMVAL(DISTANCE(CNT2, A));
(D39)                     22.18041
```

Finally let us check for the sphere in CNT1 whether it actually fulfills the conditions of the problem.

```
(C40) NUMVAL(DISTANCE(CNT1, B));
(D40)                      11.5125997

(C41) NUMVAL(DISTANCEFROMPOINTTOPLANE(CNT1, PL1));
(D41)                      11.5125996

(C42) NUMVAL(DISTANCEFROMPOINTTOPLANE(CNT1, PL2));
(D42)                      11.5125997
```

Yes, it does so! Also we have good reason to be happy with the numerical accuracy of the answer. Notice the use of symbolic evaluation in the commands (C28) through (C32).

## CONCLUSION

The foregoing routines are useful for interactive calculations of three dimensional linear structures. They could provide a model for practical interactive systems for architects and other designers, which could be enhanced by the addition of graphic facilities. Also they show how naturally vector calculus can be expressed in MACSYMA.

It is plain that the same approach can be used to express a lot more of vector calculus in MACSYMA. Linear transformations and the like can be expressed most easily. Our use of SOLVE could have been handled also by LINSOLVE. But SOLVE can also be used for problems involving curved surfaces. Differential geometry can be readily treated in this manner too, using also the MACSYMA functions for differentiating and integrating.

Textbook problems in dynamics of solid bodies are typically expressed in the language of vector calculus. Thus they can be naturally treated using this approach. A fun project would be to work out a course in rational mechanics with MACSYMA by using also its ability to solve differential equations.

BLANK PAGE

BLANK PAGE

# Varieties of Operator Manipulation [1]

Alexander Doohovskoy

Laboratory for Computer Science

Massachusetts Institute of Technology

## SUMMARY

Symbolic operator manipulation began when program (d,differentiate,verb) was perceived as data (D,Derivative,noun). Although this realization took more than 100 years (ref. 1), the nineteenth century mathphysicists soon developed this perception in three major directions: direct and indirect methods for the solution of differential equations, calculus of finite differences, and the fractional calculus.

We propose a change in MACSYMA syntax in order to accommodate the operator manipulations necessary to implement these classical symbolic methods as well as their modern counterparts. To illustrate the virtue and convenience of this syntax extension, we show how MACSYMA's pattern-matching capacity can be used to implement a particular set of operator identities due to Hirota which can be used to obtain exact solutions to nonlinear differential equations.

## 1 INTRODUCTION

What is an operator calculus? The usual technical meaning involves an isomorphism between an algebra of functions, say of the form

$$f(x) = \sum a_k x^k$$

and an algebra of operators

$$f(X) = \sum a_k X^k$$

such that pointwise multiplication of functions goes into operator multiplication:

$$f(x)g(x) \text{ -----> } f(X)g(X)$$

---

[1] This work was supported, in part, by the United States Energy Research and Development Administration under Contract Number E(11-1)-3070 and by the National Aeronautics and Space Administration under Grant NSG 1323.

The isomorphism $f(x)$ -----> $f(X)$ is also required to be linear. [1]

Basically, this means that expressions involving the operator $X$ can be manipulated algebraically. Operator algebra thus becomes a tool for finding solutions to equations or studying their structure. For example, consider the (linear) differential equation

$$(p(D))(f(t)) = g(t)$$

where $p(D)$ is a polynomial in the operator $D = d/dt$ over the coefficient ring $K[t]$ . We might try to solve this equation using various transform methods, for example, using the Laplace transform. This is the typical "indirect method" which consists of translating the original problem into a corresponding problem in some "image space", solving there, and then transforming back. If $g(t) = \exp(t^2)$,however, the Laplace transform of the RHS does not exist. The "direct" method,on the other hand, deals with the original problem itself; one could consider a factorization of the operator polynomial

$$(D-r_1)^{a1}\_(D-r_n)^{an}(f(t)) = g(t)$$

and then return the answer in the form

$$f(t) = (D-r_1)^{-a1}\_(D-r_n)^{-an}(g(t)).$$

The problem now is to give meaning to the inverse operators while preserving basic algebraic laws such as:

$$D^pD^q = D^{p+q}$$

Using a slightly different language, one can view the evolution of "operator techniques" [2] as the realization that something conceptually and computationally useful can be gained from imposing and studying the structure of the dual algebra $A^*$ of operators or functionals acting on some given algebra $A$. For example, $A$ might be $Q[x]$, the ring of univariate polynomials over the rationals. Typically, one introduces a pairing

$$< , >: A^* \times A \text{ ------> } R$$

where $R$ is some relevant ring of scalars. The next step is to define a product in the dual algebra. There are various ways of doing this; one example is

$$<L_1L_2,v> = \sum \mathrm{bin}(n,k) <L_1,x^k> <L_2,x^{n-k}> \qquad (1)$$

The product is commutative and associative. The "evaluation" map (usually called the augmentation)

$$<\epsilon,p(x)> = p(0)$$

serves as the multiplicative identity in the dual algebra of functionals acting on univariate polynomials

---

[1] In some cases functional composition is also preserved under the map.

[2] Also known as symbolic methods, symbolic calculus, functional calculus, operator calculus,operational calculus, functional operations

$$L\varepsilon = \varepsilon.L = L$$

with the product defined as in (1) above. For generalized functions with the pairing given by

$$<F,g> = \int F(x)g(x)\,dx$$

and with a product defined by convolution

$$<F*G,h) = <F(x),<G(y),h(x+y)>>$$

the role of the identity is played by the delta function.

The duality between the algebra and the functionals acting on it is made more explicit by defining the adjoint $L^*$ such that

$$<M,L^*v> = <LM,v>$$

It is also possible to contemplate the meaning of operations applied to operators, such as the derivative of an operator (element of the dual algebra). Suppose that A is the algebra of polynomials in one variable, then one meaning (refs. 2, 3) is given by

$$<L', p(x) > = <L, xp(x)>$$

A more familiar meaning of the derivative of an operator is found in the context of generalized functions (functionals) F acting on a suitable space of test functions, $\phi(t)$. The pairing is given by

$$<F,\phi(t)> = \int F(t)\,\phi(t)$$

and in this case the derivative of the functional F is defined by

$$<F',\phi(t)> = - <F,\phi'(t)>$$

(to arrive at this one uses integration by parts and then forgets). Of course the great virtue of this definition is that the meaning of F' no longer depends on the meaning or existence of a derivative (in the ordinary sense) for F. This is very convenient for functionals F which are defined as a limit of a sequence of functions. Thus, the well-known delta function(al) has a derivative which behaves as

$$<\delta'(t-a), \phi(t)> = - \phi(a)$$

These are just some of the mathematical parallels between "operator" methods applied to the difference calculus as well as the differential calculus. Rota (ref. 3) has refined the essence of these ideas into a very general theory of operators which for example finally explains the somewhat mysterious umbral operator calculus developed in classical invariant theory. In addition, it provides a neat solution to the problem of computing "connection" coefficients between various classes of polynomials.

In what follows we attempt to illustrate the variety of applications and some of the common themes in various operator calculi arising in pure and applied mathematics. MACSYMA's pattern-matching facility, together with the extended syntax we propose, is ideal for implementing these ideas.

## *2 OPERATORS IN MACSYMA*

Let us examine some of the MACSYMA programming aspects of operator algebra. For example, suppose we are dealing with a linear operator, L. In MACSYMA, there are several ways of expressing identities involving the operator L. In order to say that L is linear, we must first define a predicate to recognize sums:

```
SUMM(X):=IS(PART(X,0)="+")
```

We can then define a simplification rule by

```
LET(L(SUM),L(FIRST(SUM)) + L(REST(SUM)), SUMM,SUM)
```

An alternative method is to set up a rule using MATCHDECLARE and DEFRULE; in order to have the identity applied automatically, one can use TELLSIMP. Or, finally, one can simply say

```
DECLARE(L,LINEAR)
```

The exigencies of these methods can be overcome with a little help from primers, advisors, etc. (refs. 4, 5). Of course the last method is a response to the programming inconveniences of the first two and also attests to the mathematical importance of the notion of LINEARity. Other basic algebraic properties of operators and functions which have been subsumed under the DECLARE function include COMMUTATIVE,R-ASSOCIATIVE,L-ASSOCIATIVE,... As an example, the following MACSYMA command

```
(C33) DECLARE(L,LINEAR,M,COMMUTATIVE);
(D33)                                  DONE
```

has the following effects

```
(C34) L(X+Y);
(D34)                           L(Y) + L(X)


(C36) M(X,Y)-M(Y,X) + M(X,Y,Z) - M(Y,Z,X);
(D36)                                   0
```

Now consider the following simple identity

$$E^{a_i}p(x) = p(x + a_i) \tag{1}$$

defining a (linear) shift endomorphism $E^{a_i}$ on the algebra of univariate polynomials (over some convenient ring). How could we express this identity in MACSYMA? The problem is that we can't even write down the left-hand side of (1):

```
(C2) (E^(A[I]))(P(X));
 A
  I
E
NOT A PROPER FUNCTION - MQAPPLY
```

The usual suggestion is to break up the operator $E^{a_i}$ and append a,i as a new operands to a function E defined by

```
E(p,x,a,i):= p(x+a[i])
```

This has the unpleasant semantic consequence of destroying (at the user level) the unity and identity of the operator $E^{a_i}$ and introduces an unnecessary syntactic restriction upon a,i (recursively) forcing them to be atoms since they now appear as formal parameters in a function definition. But we may not want to apply the operator immediately. Perhaps a little simplification

$$(E^a(E^b(p)) = (E^a * E^b)(p) = (E^{a+b})(p)$$

will reveal the structure of interest to the user. That is, we may want to look at the consequences of the R-module structure given by

$$E^a(p+q) = E^a(p) + E^a(q)$$

$$(E^a+E^b)(p) = E^a(p) + E^b(p)$$

$$(E^a * E^b)(p) = E^a(E^b(p))$$

$$E^0(p) = p$$

This is simply an abstraction of the axioms for a vector space over a field in which the "scalars" are allowed to be elements of a ring R.[1]

*It is this interplay between different algebraic structures which leads to the mathematical power of operator calculi and to the programming difficulties in their implementations.*

*To take full advantage of a calculus of operators acting on some domain, one must respect the algebraic structure of BOTH the operators and the domain.*

How can we enable the MACSYMA user to use compound expressions in the functional position? In the current MACSYMA evaluation scheme, when a compound expression occurs in the functional position and is not an atom or a subscripted function, MACSYMA errs out with the message as in the example above. Instead, it is not unreasonable to return the original form with the compound expression in the functional position simply appended before the given arguments (with an "MQAPPLY"). With this modification the following kinds of expressions become possible in MACSYMA.

---

[1] In our example R is the associative ring of shift operators $E^a$.

### *2.1 Combinatorial Actions*

Numbers and lists of them can act as operators:

```
(C1)  (1)(F);
(D1)                                   1(F)  2

(C3) [1,2,3,3](F);
(D3)                   [1, 2, 3, 3](F)
```

These two examples suggest that the user can use the new operator syntax to conveniently define the action of combinatorial objects. For example, in the study of the representations of the symmetric group, [1,2,3,3] might represent the cycle structure of a conjugacy class. Many other interesting discrete actions arise from classical invariant theory, differential geometry, and the difference calculus.

### *2.2 Identities for Nonatomic Operators*

Consider now the iterates of a class of linear operators indexed in some way:

$$L[X]^N$$

We would like to say that all these are linear. One could of course DECLARE(L[X],LINEAR)[3] and induce linearity for all the iterates. With symbolic exponents however, this is not possible. Using the new syntax, we may proceed as follows:

```
(C61)  MATCHDECLARE(NNN,TRUE)$
(C62)  MATCHDECLARE(UUU,TRUE)$
(C63)  TELLSIMP((L[UUU]^NNN)(SUM),
                    (L[UUU]^NNN)(FIRST(SUM))+(L[UUU]^NNN)(REST(SUM)))$
```

Then, as a result, we obtain the following automatic simplifications:

```
(C64)  (L[U]^N)(X+L[U](Z+(L[V]^3)(A+B)));

           N          N                N       3              N       3
(D64)    (L )(X) + (L )(L  (Z)) + (L )(L  ((L )(B))) + (L )(L  ((L )(A)))
           U          U   U            U   U   V              U   U   V
```

---

[2] In future MACSYMAs one may be able to give meaning to such an expression directly through a function definition.

[3] If and when DECLARE takes nonatomic arguments

Below we give further examples of the new syntax, involving operator forms arising in differential calculus and in the finite difference calculus.

```
(C2) MATRIX([D[1,1],D[1,2]],[D[2,1],D[2,2]]);
                                  [ D      D     ]
                                  [  1, 1   1, 2 ]
(D2)                              [              ]
                                  [ D      D     ]
                                  [  2, 1   2, 2 ]

(C3) DETERMINANT(%)(F);
(D3)                       (D     D      - D     D     )(F)
                             1, 1  2, 2     1, 2  2, 1


(C4) (E^A)(F);
                                        A
(D4)                                  (E )(F)

(C5) (E^(A[J]))(F);
                                        A
                                         J
(D5)                                  (E  )(F)


(C6) (1+D)(F);
(D6)                                 (D + 1)(F)

(C7) (1/(1+D))(F);
                                        1
(D7)                                 (-----)(F)
                                      D + 1
(C8) (D[X] + D[T])(F,G);
(D8)                              (D  + D )(F, G)
                                    X    T

(C9) (EXP(D))(F);
                                         D
(D9)                                  (%E )(F)

(C10) FIRST(%);
                                          D
(D10)                                   %E

(C12) TAYLOR(D9,D,0,4);

                              4    3    2
                             D    D    D
(D12)                       (-- + -- + -- + D + 1)(F)
                             24   6    2
```

```
(C13)  (%E^D[X]*%E^D[T])(F);
                                          D  + D
                                           X    T
(D13)                                   (%E        )(F)
```

### 3 EVALUATION AND SIMPLIFICATION OF OPERATOR FORMS

Now that we can write down compound operator forms in MACSYMA, we are faced with the task of telling MACSYMA what they mean. One convenient way of doing this would be to attach properties to the non-atomic objects forming the operator part of an expression (the ability to attach properties to non-atomic objects will soon be available in MACSYMA). Naively, one might hope to simply write a function definition of the form

```
(D[X]*D[T])(F,G):= 2*D[X](DIFF(F,T),G)
```

or use MACSYMA's pattern-matching facilities

```
MATCHDECLARE([FFF,GGG,XXX,TTT],TRUE)$

DEFRULE(NAME1,(D[XXX]*D[TTT])(FFF,GGG),2*D[XXX](DIFF(FFF,TTT),GGG))$

TELLSIMP((D[XXX]*D[TTT])(FFF,GGG),2*D[XXX](DIFF(FFF,TTT),GGG))$
```

In either case, there are several ambiguities to be resolved.

1. How is MACSYMA to recognize instances of the LHS? What does the user mean when he types the function definition? Does the user intend to specify a relation involving fixed mathematical constants D[X],D[T] or does he intend to specify an identity involving the programming variables X,T ? When using DEFRULE, one uses MATCHDECLARE to restrict the sense of the variables used to describe the pattern.

2. Even if the LHS could be recognized unambiguously, the user may still be forced to label his "simplification" rules since the same LHS may transform to distinct RHS's. For example,

```
                             | 2*D[X](DIFF(F,T),F)
(D[X]*D[T])(F,F):=          <
                             | 2*D[T](DIFF(F,X),F)
```

or,

```
                                 | (...g(op1,op2)...)(op)(f)
(...(op1+op2)...)(op)(f)  --->  <
                                 | (....(op1+op2)...)(eval op f)
```

The last example reflects the possibility of making choices involving the order of simplification and evaluation.

These choices arise because we may have a relatively complicated (R-module) interaction between the algebraic structures of the operators and the elements of the domain upon which they act.

If one views the world of (algebraic/MACSYMA) expressions as made up of operators/programs in *Op* applied to objects/data in *Dom*, then the intertwining of simplification and evaluation can be represented/defined by the diagram

$$\begin{tikzcd}
\mathrm{Op} \times \mathrm{Dom} \arrow[r, "(\mathrm{simp},1)"] \arrow[d, "\mathrm{evaluation}"'] & \mathrm{Op} \times \mathrm{Dom} \arrow[d, "\mathrm{evaluation}"] \\
\mathrm{Dom} \arrow[r, "\mathrm{simp}"'] & \mathrm{Dom}
\end{tikzcd}$$

which sometimes commutes:

$$\begin{tikzcd}
(E^a E^b)(f) \arrow[r, "(\mathrm{simp},1)"] \arrow[d, "\mathrm{evaluation}"'] & (E^{a+b})(f) \arrow[dd, "\mathrm{evaluation}"{pos=0.25}, "\mathrm{evaluation}"{pos=0.75}] \\
E^a(f(x+b)) \arrow[d, "\mathrm{evaluation}"'] & \\
f(x+b+a) \arrow[r, "\mathrm{simp}"'] & f(x+(a+b))
\end{tikzcd}$$

and sometimes does not commute:

$$\begin{tikzcd}
(D\,D^{-1})f \arrow[r, dashed, "(\mathrm{simp},1)"] \arrow[d, dash, "\mathrm{evaluation}"'] & (D^{-1}D)f \arrow[d, dash, "\mathrm{evaluation}"] \\
D^{-1}(Df) = \int_0^t f(s)ds \arrow[d, "\mathrm{evaluation}"'] & D(D^{-1}f) = D\int_0^t f(s)ds \arrow[d, "\mathrm{evaluation}"] \\
f(t) - f(0) \arrow[r, dashed, "//" marking, "\mathrm{simp}"'] & f(t)
\end{tikzcd}$$

which says that

$$DD^{-1}f \neq D^{-1}Df$$

It is clear that this noncommutativity is an impediment to the development of an operational calculus.

Historically, there have been several approaches to the restoration of commutativity in the above diagram. One method is conceptually trivial. The diagram can be made commutative by redefining the operand

$$f(t) \dashrightarrow f(t) - f(0)$$

to have vanishing initial condition. One can also define the inverse indefinitely by

$$D^{-1} = \int^{x} + c = \int_{a}^{x}$$

and treat the constants separately. This leads to the symbolic calculus systematically developed by Murphy,Carmichael,Hargreave,Boole (ref. 6, 7) and others. Together with the Leibniz rule for products and the Taylor expansion theorem, the principal identities are (ref. 8)

$$F(D)\, e^{g(x)} = e^{g(x)}\, F(D + g'(x))$$

$$F(x + g'(D))\, e^{g(D)} = e^{g(D)}\, F(x)$$

$$F(D^2)\ \sin|\cos\ (a\ x) = F(-a^2)\ \sin|\cos\ (a\ x)$$

$$F(D^2)\ \sinh|\cosh\ (a\ x) = F(-a^2)\ \sinh|\cosh\ (a\ x)$$

Using the extended operator syntax suggested here, one can easily implement these identities and apply them to the solution of differential equations. We illustrate below some of the symbolic methods which can be used to deal with ordinary and partial differential equations. One advantage of these "direct" methods as opposed to "indirect" transform methods is the minimization of existence assumptions.

### *4.1 Ordinary Differential Equations: Constant Coefficients*

There are several methods available in MACSYMA to solve differential equations (refs. 9,10). In this section we discuss the "direct" symbolic method applied to ordinary differential equations with constant coefficients.

Let D be differentiation with respect to t and consider the differential equation

$$(D + 1)f = t^3$$

An operator approach to the solution gives

$$f = (1+D)^{-1}t^3$$

$$f = (1 - D + D^2 - D^3 + \ldots)t^3$$

The action of the operators[1]

$$f = t^3 - Dt^3 + D^2t^3 - D^3t^3 + \_$$

$$f = t^3 - 3t^2 + 6t - 6$$

(often) yields substantial dividends by clarifying the structure of the problem and providing effective means of computation.

Essentially we have used a Euclidean identity

$$P(D)Q(D) + R(D) = 1$$

applied to the given function $g(t) = t^3$

$$[P(D)Q(D) + R(D)]g = g \text{ -----> } P(D)Q(D)g = g$$

since we arrange $R(D)g = 0$ (by making the degree R in D high enough). We can then pick out our solution as $f = Q(D)g$.

Now consider a slightly more general differential equation $P(D)f = g$ (constant coefficients) where g may not be a simple polynomial. One can still look for f directly by inverting P(D)

$$f = P^{-1}(D)g$$

but the RHS may not be compactly expressible now. To remedy this one can generalize the previous idea and look for a Q(D) such that

$$Q(D)g = 0.$$ [2]

Then using the extended Euclidean algorithm to look for A(D), B(D) such that

$$P(D)A(D) + B(D)Q(D) = U(D)$$

one hopes that U(D) will be 1. If it is, then

$$[P(D)A(D) + B(D)Q(D)]\, g = g \,;\, P(D)A(D)\, g = g$$

and we can pick out the solution as $f = A(D)g$.

If $U(D) \neq 1$, then

$$[P(D)A(D) + B(D)Q(D)]\, g = U(D)\, g \,;\, P(D)\, U^{-1}(D)A(D)\, g = g$$

and we can again pick out our solution as $f = U^{-1}(D)A(D)\, g$ hoping that the lower degree of U(D) will make it easier to invert than P(D). A(D)g may or may not be simpler than the original g to deal with.

---

[1] This simple example is intended only to help specify the issue of interaction between the operator algebra and the module of functions

[2] This statement (due to Robert Feinberg) formalizes what one does intuitively when solving equations by "inspection"

Of course, independent of the Euclidean algorithm, one might try to find an operator $L_2$ such that an $L_1$ can be found with the property that

$$P(D)L_1 + L_2 = I$$

Then the solution can be obtained as above.

### *4.2 Linear Equations with Variable Coefficients*

As an example of the economy sometimes afforded by working directly with the differential operators, consider the following equation (ref. 11)

$$(D^4 + 2x^{-1}D^3 - x^{-2}D^2 + 2x^{-3}D - 1)(f) = 0$$

One can attempt a power series solution to this equation (ref. 10); but another approach is to factor the differential operator as

$$(D^2 + x^{-1}D + 1)(D^2 + x^{-1}D - 1)(f) = 0$$

Since the two factors commute, one can find a solution of the form $f = f_1 + f_2$ where

$$(D^2 + x^{-1}D + 1)(f_1) = 0 \text{ , } (D^2 + x^{-1}D - 1)(f_2) = 0$$

These simpler Bessel equations then lead to the solution of the original problem:

$$f = c_1J_0(x) + c_2Y_0(x) + c_3J_0(ix) + c_4Y_0(ix)$$

Thus, by taking advantage of the operator algebra instead of using brute force, one can discover or preserve the inherent structure of a problem. Moses (ref. 12) has recently elucidated this idea for algebraic algorithms; it applies equally well to operational methods in applied mathematics.

### *4.3 Linear Partial Differential Equations*

The principal operator identities above have their natural analogues in the multivariable case. Let $D_1$, $D_2$ denote the partial derivatives with respect to x,y. Then

$$F(D_1,D_2)\, e^{f(x,y)} = e^{f(x,y)}\, F(D_1+D_1f, D_2+D_2y)$$

$$e^{f(D_1,D_2)}\, F(x,y) = F(x+D_1f, y+D_2f)\, e^{f(D_1,D_2)}$$

$$F(D_1,D_2)\, f(ax + by) = F(a,b)\, f^{(n)}(ax + by)$$

As an example (ref. 13) consider the initial-value problem

$$u_t = au_x + bu_y + cu_z \text{ ; } t > 0$$

with initial values $u(x,y,z,0+) = f(x,y,z)$. Let $D^{-1}$ denote the operator

$$D^{-1}\, g(x,y,z,t) = \int_0^t g(x,y,z,s)\, ds$$

Then the PDE can be integrated with respect to t and expressed in the form

$$u(x,y,z,t) - u(x,y,z,0+) = Q\,(aD_x + bD_y + cD_z)\, u(x,y,z,t)$$

$$u(x,y,z,t) - f(x,y,z) = Q\,(aD_x + bD_y + cD_z)\, u(x,y,z,t)$$

or, using the direct symbolic method to solve for u, we obtain

$$\begin{aligned} u(x,y,z,t) &= [1 + Q\,(aD_x + bD_y + cD_z)]^{-1} f(x,y,z) \\ &= \sum (-1)^n Q^n (aD_x + bD_y + cD_z)^n\, f(x,y,z) \\ &= \sum (-1)^n (t^n/n)\, (aD_x + bD_y + cD_z)^n\, f(x,y,z) \\ &= e^{-t(aD_x + bD_y + cD_z)}\, f(x,y,z) \\ &= e^{-atD_x} e^{-btD_y} e^{-ctD_z}\, f(x,y,z) \\ &= f(x-at, y-bt, z-ct) \end{aligned}$$

using Taylor's theorem in operator form.

This example again illustrates the power and the economy of the symbolic method which takes advantage of the inherent algebraic structure of the problem and returns a more meaningful result.

### *4.4 Nonlinear Partial Differential Equations*

Recently, in looking for exact solutions to nonlinear evolution equations, Ryogo Hirota (refs. 14, 15) has developed a calculus based on the differential operator[3]

```
                    N                          N            |
(***)             (D )(A, B) = ((DIFF  - DIFF ) )(A(T)*B(S))|
                    T                T       S              |S=T
```

Using an appropriate substitution, one can express a given equation in terms of such differential operators. The resulting forms are then amenable to a perturbation expansion which leads to the solution.

Using this approach Hirota has been able to treat the modified Korteweg-deVries equation,the nonlinear Schrodinger equation,wave-wave interactions,the two-dimensional K-dV equations,and the two-dimensional sine-Gordon equation.

The differential operators (***) satisfy a number of identities which are used to replace the usual partial derivatives with bilinear forms involving the new differential operators. For example,

---

[3] We use $DIFF_X$ to denote the partial derivative

```
                                             PP1
                              PP1            d  AA1
(D17)                       (D   )(AA1, 1) = -------
                              XX1                PP1
                                            dXX1


                    PF1                   PF1   PF1
(D18)             (D  )(AA1, BB1) = (- 1)     (D  )(BB1, AA1)
                    XX1                         XX1


          CC1 D
               XX1
(D31)  (%E         )(AA1(XX1), BB1(XX1)) = BB1(XX1 - CC1) AA1(XX1 + CC1)


                                          CC1 D
                  CC1 DIFF                     XX1
                           XX1 BB1      (%E       )(BB1, AA1)
(D41)           (%E           )(---) = ------------------------
                               AA1    COSH(CC1 D   )(AA1, AA1)
                                                XX1


                                     D   (BB1, AA1)
                            BB1       XX1
(D42)                      (---)   = --------------
                            AA1              2
                                XX1       AA1


                              2                     2
                            (D   )(BB1, AA1)  BB1 (D   )(AA1, AA1)
             BB1              XX1                   XX1
(D44)        (---)         = ---------------- - --------------------
             AA1                     2                   3
                 XX1 XX1          AA1                 AA1


                                           2
                                         (D   )(AA1, AA1)
                                          XX1
(D47)                 LOG(AA1)         = ----------------
                               XX1 XX1             2
                                             2 AA1
```

As an example of the Hirota method, consider the two-wave interaction described by the equations

```
(D101)                          F1  V1 + F1   = - F1 F2
                                  X        T

(D102)                           F2  V2 + F2   = F1 F2
                                   X        T
```

where the waves F1 and F2 propagate with velocities V1 and V2. The substitution

```
(C103) EV(D101,F1=G1/F,F2=G2/F);
                               G1          G1         G1 G2
(D103)                        (--)  V1 + (--)   = - -----
                               F           F             2
                                  X           T         F

(C105) EV(D102,F1=G1/F,F2=G2/F);
                               G2          G2       G1 G2
(D105)                        (--)  V2 + (--)   = -----
                               F           F          2
                                  X           T      F
```

yields the equations (using D42 here)

```
(C106)  APPLY1(D103,RULEH71);
                          V1 D (G1, F)    D (G1, F)
                               X            T              G1 G2
(D106)                    ------------ + --------- = - -----
                                2             2              2
                               F             F              F

(C107)  APPLY1(D105,RULEH71);
                           V2 D (G2, F)    D (G2, F)
                                X            T           G1 G2
(D107)                     ------------ + --------- = -----
                                 2             2           2
                                F             F           F
```

Hirota now uses a perturbation analysis

```
                              5          4          3          2
(D108)     F = F  EPS  + F  EPS  + F  EPS  + F  EPS  + F  EPS + F
                5          4          3          2          1      0

                                5          4          3          2
(D109)       G1 = G  EPS  + G  EPS  + G  EPS  + G  EPS  + G  EPS
                   5          4          3          2          1
```

(D110)
$$G2 = H_5 \, EPS^5 + H_4 \, EPS^4 + H_3 \, EPS^3 + H_2 \, EPS^2 + H_1 \, EPS$$

Upon substituting and equating like powers of $\epsilon$, one obtains the following equations in the first few orders of $\epsilon$:

$$(D_T + V1 \, D_X)(G_1,1) = 0 \, , \, (D_T + V2 \, D_X)(H_1,1) = 0$$

$$(D_T + V1 \, D_X)(G_1,F_1) + (D_T + V1 \, D_X)(G_2,1) = 0$$

$$(D_T + V2 \, D_X)(H_1,F_1) + (D_T + V2 \, D_X)(H_2,1) = 0$$

Surprisingly, the zeroth order solutions

$$G_1 = G_1(x-v_1 t) \, ; \, H_1 = H_1(x-v_2 t)$$

induce an exact solution in a relatively simple way. All the higher order equations are automatically satisfied if all the higher order terms are chosen to be zero and $f_1$ satisfies the equations

(D112)
$$F1_X \, V1 + F1_T = H_1$$

(D113)
$$F2_X \, V2 + F2_T = -G_1$$

These have a general solution

$$F_1 = U_1(X - V1 \, T) + U_2(X - V2 \, T)$$

where
$$(DIFF_T + V1 \, DIFF_X)U_2(X - V2 \, T) = H_1(X - V2 \, T)$$

$$(DIFF_T + V2 \, DIFF_X)U_1(X - V1 \, T) = - \, G_1(X - V1 \, T)$$

and leads to the exact solution of the original equation

$$F1 = - \frac{(V2 \, DIFF_X + DIFF_T)(F_1(X - T \, V1))}{F_2(X - T \, V2) + F_1(X - T \, V1)}$$

$$F2 = - \frac{(V1 \, DIFF_X + DIFF_T)(F_2(X - T \, V2))}{F_2(X - T \, V2) + F_1(X - T \, V1)}$$

# REFERENCES

1. Arbogast,L.F.A.: Du Calcul des Derivations. Strasbourg, 1800.

2. Pincherle,S.: Operatori lineari e coefficienti di fattoriali. Alti Accad. Naz. Lincei, Rend. Cl. Fis. Mat. Nat. (6) XVIII, 1933, pp. 417-519.

3. Rota,G.: Finite Operator Calculus. J. Math. Analysis and Applications, Vol. 42, No. 3, Academic Press, June 1973.

4. Genesereth,M.R.: An Automated Consultant for MACSYMA. Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper no. 30 of this compilation.)

5. Lewis,V.E.: User Aids for MACSYMA. Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper no. 28 of this compilation.)

6. Boole,A.: Treatise on Differential Equations. Chelsea,New York (reprinted from 1859).

7. Boole,A.: Treatise on the Calculus of Finite Differences. Dover,New York, 1960 (reprinted from 1872).

8. Stephens,E.: Elementary Theory of Operational Mathematics. McGraw-Hill Book Co., 1937.

9. Golden,J.P.: The Evaluation of Atomic Variables in MACSYMA. Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper no. 12 of this compilation.)

10. Lafferty,E.L.: Power Series Solution of Ordinary Differential Equations. Proceedings of the 1977 MACSYMA Users' Conference, NASA CP-2012, 1977. (Paper no. 34 of this compilation.)

11. Martin,W.T.;and Reissner,E.: Elementary Differential Equations. Addison-Wesley Pub. Co., 1964.

12. Moses,J.: Algebraic Structures and Their Algorithms. in Algorithms and Complexity (J.F. Traub ed.), Academic Press, 1976.

13. Liverman,T.P.G.: Generalized Functions and Direct Operational Methods. Prentice-Hall Inc., 1964.

14. Hirota,R.: Direct Method of Finding Exact Solutions of Nonlinear Evolution Equations. in Backlund Transformations, the Inverse Scattering Method, Solitions, and Their Applications (R.M.Miura ed.), Springer-Verlag, 1976.

15. Hirota,R.: A New Form of Backlund Transformations and Its Relation to the Inverse Scattering Problem. Progress of Theoretical Physics, Vol. 52, No. 5, Nov. 1976, pp. 1498.

# PROGRESS REPORT ON THE DETERMINANT OF A RANDOM MATRIX

F. A. Grünbaum
University of California, Berkeley

## ABSTRACT

Let $X_1 \ldots X_n$ denote a random vector with Gaussian distribution with mean vector $\bar{m}_i$ and correlation matrix $R_{ij}$.

The explicit computation of moments of the type

$$E(X_1^{p_1} X_2^{p_2} \ldots X_n^{p_n}) \tag{1}$$

is best done by expressing the usual powers in terms of Hermite polynomials $H_n(x)$ and computing the expectations for these in terms of multigraphs. (See ref. 1.) Computations similar to these are common in quantum field theory where: $\phi^n : = H_n(\phi)$.

Here we propose to describe the use of MACSYMA for dealing with a much tougher but related problem, described below.

If $A$ is an $n \times n$ real matrix we want to find out what information about $A$ is contained in the set of moments of the random variable.

$$\det(A + E) \tag{2}$$

Here $E$ denotes an $n \times n$ matrix each of whose entries is a Gaussian random

variable with mean zero and some joint correlation matrix.

In the case of independent entries with a common non-zero variance the result -- partially obtained using MACSYMA is

Theorem. The moments of $\det(A + E)$ determine exactly the singular values of $A$ and its determinant.

Crucial for this work is the possibility of computing quantities similar to (1) where powers of $X_i$ are replaced by powers of minors of the matrix $E$. We obtain some interesting multigraph expansions but the picture is still far from complete and a good deal of extra experimentation is needed. We anticipate the MACSYMA will be quite valuable in this aspect of our work.

## REFERENCE

1. Grünbaum, F. A.: Inverse Problems for Nonlinear Random Systems. Partial Differential Equations and Related Topics, Volume 446 of Lectures Notes in Mathematics, J. Goldstein, ed., Springer-Verlag, 1975, pp. 247-263.

# BIBLIOGRAPHY

Ament, W.S.: Intensity Statistics for a Multiple-Scatter Model Via Computer Symbol Manipulation. Naval Research Laboratory, Code 5404, Washington, D. C.

Ananda, M.P.; Broucke, R.A.; and Hamata, N.: Computation of Force Components of a Gravitational Potential. Tech. Memo. 391-444, Jet Propul. Lab., California Inst. Technol., June 1973.

Andersen, C.M.; and Bowen, J.T.: A Computer Program for Anisotropic Shallow Shell Finite Elements Using Symbolic Integration. NASA TM X-3325, 1976.

Andersen, C. M.; and Darden, G. C.: Evaluation of Integrals for a Ten-Node Isoparametric Tetrahedral Finite Element. Paper presented at the SIAM-SIGNUM Fall Meeting (San Francisco), Dec. 1975.

Andersen, C.M.; and Noor, Ahmed K.: Use of Group-Theoretic Methods in the Development of Nonlinear Shell Finite Elements. Proceedings of the Conference on Symmetry, Similarity, and Group Theoretic Methods in Mechanics, Univ. of Calgary (Alberta, Canada), Aug. 1974.

Andersen, C.M.; and Noor, Ahmed K.: A Computerized Symbolic Integration Technique for Development of Triangular and Quadrilateral Composite Shallow Shell Finite Elements. NASA TN D-8067, 1975.

Andersen, C.M.; and Noor, Ahmed K.: Free Vibrations of Laminated Composite Elliptic Plates. Adv. in Engr. Sci., vol. 2, NASA CP-2001, 1976, pp. 425-438.

Barouch, E.; and Kaufman, G.M.: Estimation of Undiscovered Oil and Gas. NSF Grant SIA74-22773, Sloan School, Massachusetts Inst. Technol., 1976.

Barton, D. R.; and Zippel, R.: A Polynomial Decomposition Algorithm, 1976 Proceedings of Symposium on Symbolic and Algebraic Computation, Assoc. Comput. Mach., August 1976, pp. 356-358.

Bender, C.M.; Keener, R.W.; and Zippel, R.E.: New Approach to the Calculation of $F[1](\alpha)$ in Massless Quantum Electrodynamics. Phys. Rev. (to appear), NSF Grant 29463, ERDA Contract E(11-1)-3070, and Alfred P. Sloan and NSF fellowships.

Bers, A.: Symbolic Computation of Nonlinear Wave Interactions. NSF(GK037979X1) and ERDA (AT(11-1)-3037), Bull. of Am. Phys. Soc., Mar. 1975.

Bers, A.; Kulp, J.L.; Watson, D.C.: Analytic Studies of Nonlinear Plasma Problems Symbolic Manipulation Programs on a Computer. Massachusetts Instit. Technol., Res. Lab. of Electronics, Quar. Prog. Rep. no. 108, Jan. 1973, pp. 167-185.

Bers, A.; and Reiman, A.: Nonlinear Interaction of Three Wave Packets in Time and Space. Massachusetts Instit. Technol. Research Laboratory of Electronics, AEC Contract AT(11-1)-3070, Plasma Research Report, PRR-757, Apr. 1975.

Blau, Irwin: The Value of Information in Non-Zero/Sum Stochastic Games. Ph.D. Thesis, Massachusetts Instit. Technol., May 1974.

Bogen, R.A.: Automatic Computation of Direct and Inverse Laplace Transforms Using Computer Symbolic Mathematics. Univ. of Hawaii, ONR Contract N00014-70-A-0362-0006 and by NSF Grant MCS75-22893, 1976.

Bogen, R.A.; et al. : The MACSYMA Reference Manual, Version 8, ONR Contract N00014-75-C-0661, Lab. for Comp. Sci., Massachusetts Inst. Technol., Nov. 1975.

Bogen, R. A.; and Pavelle, R.: Indicial Tensor Manipulation on MACSYMA. Lett. Math. Phys., vol. 2, 1977, in press.

Caviness, B. F.; and Fateman, R.J.: Simplification of Radical Expressions, Proceedings of the Symposium on Symbolic and Algebraic Computation, Assoc. Comput. Mach., Aug 1976, pp. 329-338.

Chattergy, R.: Reliability Analysis of On-Line Computer Systems Using Computer-Algebraic Manipulations. ONR contract N00014-70-A-0362-0001 and NASA contract NAS2-8590, Univ. of Hawaii, Aloha System Technical Report A76-1, Jan 1976.

Chow, S.K.; Hou, A.H.; and Landweber, L.: Hydrodynamic Force and Moment Coefficients of an Elongated Body of Revolution Rapidly Approaching a Free Surface in Planar Motion. Westinghouse Research Laboratories, Pittsburgh, PA., Research Report 75-1G0-LANCH-P1 [ES 575], Aug. 1975.

Chow, S.K.; Hou, A.H.; and Landweber, L.: Hydrodynamic Coefficients of an Elongated Body Rapidly Approaching a Free Surface. Journal of Hydronautics, vol. 10, no. 2, Apr. 1976.

Chu, D.: Machine Inversion of Remote Sensing Data. SB Thesis, Massachusetts Instit. Technol., June 1975.

Cohen, J.: Symbolic and Numerical Computer Analysis of the Combined Local and Overall Buckling of Rectangular Thin-Walled Columns. Comp. Meth. in Appl. Mech. and Engineering, vol. 7, 1976, pp. 17-38.

Dagalakis, N.: Electromechanical Design and an Inertial Spool Superconducting Generator. Ph.D. Thesis, Massachusetts Instit. Technol., Nov. 1974.

Djaferis, T.E.; and Mitter, S.K.: Exact Solution of Lyapunov's Equation Using Algebraic Methods. Proceedings of the Decision and Control Conference, (Clearwater Beach), Fla., Dec. 1976.

Djaferis, T.E.; and Mitter, S.K.: Exact Solution of Some Linear Matrix Equations Using Algebraic Methods. ERDA grant D(49-18)-2087 and NASA grant NGL-22-009-124, Electonic Sys. Lab., Massachusetts Inst. Technol., ESL-P-746, May 1977, 22 pp.

Fan, P.Y.: Computational Problems in Modeling the Oil and Gas Discovery Process. SM Thesis, Massachusetts Instit. Technol., June 1976.

Fateman, R.J.: Optimal Code for Serial and Parallel Computation. Communications of the Assoc. Comput. Mach., vol 12, no. 12, Dec. 1969, pp. 694-695.

Fateman, R.J.: The User-Level Semantic Matching Capability in MACSYMA. Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, Assoc. Comput. Mach., 1971, pp. 311-323.

Fateman, R.J.: A Case History in Interactive Problem Solving. SIGSAM Bulletin no. 28, Dec. 1973.

Fateman, R.J.: On the Computation of Powers of Sparse Polynomials. Stud. in Appl. Math., vol. 53, no. 2, June 1974, pp. 145-155.

Fateman, R.J.: An Algorithm for Deciding the Convergence of the Rational Iteration x[n+1]=f(x[n]). Accepted for Publication, Assoc. Comput. Mach. Trans. on Math Software.

Fateman, R.J.: The MACSYMA 'Big-Floating-Point' Arithmetic System. Proceedings of Symposium on Symbolic and Algebraic Computation, Assoc. Comput. Mach., Aug. 1976, pp. 209-213.

Fateman, R.J.: An Algorithm for Deciding the Convergence of the Rational Iteration x[n+1]=f(x[n]). Accepted for Publication, Assoc. Comput. Mach. Trans. on Math Software.

Forman, E.H.: Statistical Models and Methods for Measuring Software Reliability. Contract N00014-67-A-0214, Tech. Memo. TM-64805, George Washington Univ., Dec. 1974.

Fuller, S.H.; Gaschig, J.G.; and Gillogly, J.J.: Analysis of the Alpha-Beta Pruning Algorithm. Contract N00014-67-A-0214, Dept. of Computer Science Report, Carnegie-Mellon Univ. Pittsburgh, PA., July 1973.

Gosper, R.W.: Acceleration of Series. ONR contract no. N00014-70-A-0362-0005, Massachusetts Instit. Technol. Art. Int. Lab., AIM-304, Mar. 1974.

Goss, Blake Alan: Modulation by Mode Conversion with an Electrooptical Substrate. SB Thesis, Massachusetts Inst. Technol., May 1973.

Guibas, L.J.: The Analysis of Hashing Algorithms. Xerox Palo Alto Research Center, CSL-76-3, July 1976.

Herschcovitch, A.; and Politzer, P.A.: Time Evolution of Velocity-Space Instabilities on Counterstreaming, Magnetically Confined Electron Beams. NSF Grant ENG. 75-06242, Phys. Rev. Letters, vol. 36, no. 23, June 1976.

Hirshman, S.P.; Sigmar, D.J.; and Bers, A.: Application of MACSYMA to the Neoclassical Transport Problem of Impurities and Alpha-Particles in Tokamaks. Massachusetts Instit. Technol., Plasma Dynamics Internal Memo 18, Mar. 1974.

Howard, J.C.: The Formulation of Simulation Models of Aeronautical Systems. Ames Res. Cen., NASA, Seventh Annual Conference on Modeling and Simulation, Univ. of Pittsburgh, Apr. 1976.

Karney, C.F.F.: Parametric Coupling to Low Frequency Plasma Waves. SM Thesis, Massachusetts Inst. Technol., Jan. 1974.

Karney, C.F.F.: Stochastic Heating of Ions in a Tokamak by RF Power. PhD Thesis, Massachusetts Inst. Technol., May 1977.

Karney, C.F.F.; Bers, A; Kulp, J.L.: Parametric Excitation of Ion Waves by Waves Near the Lower Hybrid Frequency. NSF Grant (GK-28282X1), Massachusetts Instit. Technol. Quarterly Progress Report No. 110, July 1973, pp. 104-117.

Khalil, H.M., and Ulery, D.L.: Multiparameter Families of Difference Approximations to the Heat Operator in an Arbitrary Region. ONR Contract N0014-70-A-0362-0001, Adv. in Comp. Meth. for Part. Diff. Equations, AICA, 1975, pp. 304-311.

Knuth, D.E.: Notes on Generalized Dedekind Sums. NSF Grant GJ 36473, ONR Contract N00014-70-A-0362-0001, Stanford Univ., STAN-CS-75-480, Feb. 1975.

Knuth, D.E.; and Pardo, L.T.: Analysis of a Simple Factorization Algorithm. NSF Grant DCR72-03752 A02, ONR Contract NR 044-402, and IBM, Stanford Univ., STAN-CS-76-538, Jan. 1976.

Knuth, D.E.: Evaluation of Porter's Constant. NSF Grant DCR72-03752 A02, ONR N00014-70-A-0362-0001, Comp. and Math. With Appl., Feb. 1976, pp. 137-139.

Kong, J.A.: Optics of Bianisotropic Media. Proceedings of Optical Society of America, Washington, DC, April 1974.

Kong, J.A.: Analytical Studies of Electromagnetic Problems with MACSYMA. Proceedings of the Joint Conference of the Union Radio Science Internationale and the Institute of Electrical and Electronic Engineers, Atlanta, Ga., Joint Services Electronics Program Contract 8142, June 1974.

Kong, J.A.: Probing the Earth with Microwave Remote Sensing: An Example of Semi-Numerical Studies of Electromagnetic Waves by Computers. Massachusetts Instit. Technol., Res. Lab. of Electronics Progress Report 114, 1974, pp. 81-85.

Krajcik, R.A., and Nieto, M.M.: Bhabha First-Order Wave Equations. V. Indefinite Metric and Foldy-Wouthuysen Transformations. ERDA, Phys. Rev. D., vol. 14, no. 2, July 1976.

Kulp, J.L.: Use of Symbolic Computation for Nonlinear Wave Interactions. SB and SM Thesis, Massachusetts Instit. Technol., Aug. 1973.

Kulp, J.L.; Bers, A.; and Moses, J.: New Capabilities for Symbolic Computation in Plasma Physics. Proceedings of the Sixth Conference on Numerical Simulation of Plasmas, Univ. of California at Berkeley, July 1973.

Kulp, J.L., Karney, C.F.F., and Bers A.: Symbolic Computation of Nonlinear Wave-Wave Interactions. NSF Grant (GK-28282X1), Massachusetts Instit. Technol. Quar. Prog Rep. no. 110, July 1973, pp. 86-102.

Lebne-Dengel, Z.; E. G. Njoku; and J. A. Kong: A MACSYMA Study of Waves in Uniaxial Media. Massachusetts Inst. Technol., Res. Lab. of Electronics Progress Report 114, July 1974, pp. 86-87.

Lee, C.E. and Carruthers, L.M.: LARC-1: A Los Alamos Release Calculation Program for Fission Product Transport in HTGRs During the LOFC Accident. LA-NUREG-6563-MS, Los Alamos Scientific Laboratory, Nov. 1976.

Levner, D.: Mathematical Tools for Markov Modeling of System Availability and Reliability. Carnegie-Mellon Univ., Pittsburgh, Pa.

Lewis, V. E.: Introduction to ITS for the MACSYMA User. Lab. for Comp. Sci., Massachusetts Inst. Technol., 1977.

Massachusetts Instit. Technol. World Oil Project: Oil Supply Forecasting Using Disaggregated Pool Analysis. Working Paper No. MIT-EL-76-009WP, May 1976.

Metcalfe, Ralph: Spectral Methods for Boundary Value Problems. Ph.D. Thesis, Massachusetts Instit. Technol., Sept. 1973.

Moses, J.: Algebraic Simplification: A Guide for the Perplexed. Comm. Assoc. Comput. Mach., vol. 14, no. 8, Aug. 1971, pp. 527-537.

Moses, J.: Symbolic Integration: The Stormy Decade. Comm. Assoc. Comput. Mach., vol. 14, no. 8, Aug. 1971, pp. 548-560.

Moses, J.; and Yun, D.Y.Y.: The EZ GCD Algorithm. Proceedings of the Assoc. Comput. Mach. National Convention, Aug. 1973.

Moses, J.: The Evolution of Algebraic Manipulation Algorithms. IFIP 74, North-Holland Pub. Co., 1974.

Moses, J.: MACSYMA - The Fifth Year. Proceedings of the Eurosam 74 Conf., Stockholm, Aug. 1974.

Moses, J.: The Current Capabilities of the MACSYMA System. Proceedings Assoc. Comput. Mach. National Conference, Oct. 1975.

Moses, J.: Algebraic Structures and Their Algorithms. Proceedings of the Symposium on Algorithms and Complexity: New Directions and Recent Results, Pittsburgh, Apr. 1976.

Moses, J.: An Introduction to the Risch Integration Algorithm. Proceedings Assoc. Comput. Mach. National Conference, Oct. 1976.

Noor, A.K.; and Andersen, C.M.: Mixed Isoparametric Elements for Saint-Venant Torsion. Comp. Meth. Appl. Mech. and Engr., vol. 6, 1975, pp. 195-218.

Odlyzko, A.M.: Lower Bounds for Discriminants of Number Fields. Ph.D. Thesis, Massachusetts Instit. Technol., 1976.

Parrish, W.; and Pickens, J.R.: M.I.T-Mathlab meets UCSB-OLS, an Example of Resource Sharing. NIC 17161, RFC 525, June 1973.

Pavelle, R.: General Static Curved Solutions of Einstein's Equations for Spherically Symmetric Metrics. Perception Technology Corp., Winchester, Mass. ARPA Contract DAHC15 73 C 0369, 1974.

Pavelle, R.: Yang's Gravitational Field Equations. Perception Technology Corp., Winchester, Mass., Phys. Rev. Letters, vol. 33, 1974, p. 1461.

Pavelle, R.: Unphysical Solutions of Yang's Gravitational-Field Equations. Phys. Rev. Letters, vol. 34, no. 17, Apr. 1975, p. 1114.

Pavelle, R.: Conserved Vector Densities and their Curl Expressions. J. Math. Phys., vol. 16, May 1975, pp. 1199-1200.

Pavelle, R.: Multiple Covariant Differentiation - A New Method. Gen. Relativ. and Gravit., vol. 7, no. 4, 1976, pp. 383-386.

Pavelle, R.: Unphysical Characteristics of Yang's Pure-Space Equations. Phys. Rev. Letters, vol. 37, no. 15, Oct. 1976, pp. 961-964.

Perception Technology Corporation Use of Symbolic Manipulation Techniques to Examine Gravitational and Unified Field Theories. Final Technical Report, Contract No. DAHC 15 73 C 0369, June 1974.

Pless, Vera; and Sloane, N.J.A.: Complete Classification of (24,12) and (22,11) Self-Dual Codes. Lab. for Comp. Sci. TM-49, Massachusetts Inst. Technol., June 1974.

Pless, Vera; and Sloane, N.J.A.: On the classification and Enumeration of Self-Dual Codes. ARPA order no. 2095 and ONR contract no. N00014-70-A-0362-006, J. of Combin. Theory, Series A, vol. 18, no. 3, May 1975, pp. 313-335.

Ramakrishnan, R.: A Study of Pool Model Ambiguities and of the Statistics of Parameter Estimation, with an Application in Nitrogen Metabolism. Ph.D. Thesis, Columbia Univ., 1974.

Reiman, A.; and Bers, A.: Stability Analysis of a Finite Difference Scheme Using Symbolic Computation. Massachusetts Instit. Technol. Res. Lab. of Electronics, PRR-756, April 1975.

Robbins, K.A.: Disk Dynamos and Magnetic Reversal. Ph.D. Thesis, Massachusetts Instit. Technol., June 1975.

Rosen, B. and Roycraft, T.J.: Automation of Meteorological Perturbation Theory. ARPA Grant DA-ARO-D-31-124-73-G138, Stevens Inst. Technol., Prog. Rep., Dec. 1973.

Rosen, B. and Roycraft, T.J.: Finite Amplitude Baroclinic Waves. Stevens Inst. Technol., March 1976.

Rumore, F.C.: The Rotating Air Gap Armature Machine as a Superconducting Induction Motor for High Speed Ship Propulsion. SM Thesis, Massachusetts Instit. Technol. Aug. 1976.

Stoutemyer, David R.: Analytical Optimization using Algebraic Manipulation. Univ. of Hawaii, The Aloha System, TR A74-3, July 1974.

Stoutemyer, D.R.: Computer Algebraic Manipulation for the Calculus of Variations, the Maximum Principle, and Automatic Control. Univ. of Hawaii, Aloha System, TR A74-5, Nov. 1974.

Szeto, Ellen W.: Dumbbell Model for the Classical Radiation Reaction. BA Thesis, Mount Holyoke Coll., Mar. 1977.

Townsend, J.C.: Second-Order Small Disturbance Theory for Hypersonic Flow Over Power-Law Bodies. Ph.D. Thesis, Univ. of Virginia, Dec. 1974.

Trager, B.: Algorithms for Computing with Algebraic Functions. SM Thesis, Massachusetts Instit. Technol., June 1976.

Trager, B.: Algebraic Factoring and Rational Function Integration. Proceedings of the Symposium on Symbolic and Algebraic Computation, Assoc. Comput. Mach., Aug. 1976, pp. 219-226.

Underhill, Dwight W.; Reeds, James A.; and Bogen, Richard: Mathematical Analysis of Velocity Programmed Chromatography. Analytic Chem., vol. 45, Dec. 1973, p. 2314.

Walton, I.G.: The Development and Application of the Rayleigh-Ritz and Galerkin Methods. MS Thesis, Univ. of California, Santa Cruz, June 1975.

Wang, P.: Automatic Computation of Limits. Proceedings of the 2nd Symposium on Symbolic and Algebraic Manipulation., Assoc. Comput. Mach., 1971.

Wang P.: Symbolic Evaluation of Definite Integrals by Residue Theory in MACSYMA. Proceedings IFIP 74, pp. 823-827.

Wang, P.; and Rothschild, L.: Factoring Multivariate Polynomials over the Integers. Math. Comp., 1975, pp. 935-950.

Wang, P.: Factoring Multivariate Polynomials over Algebraic Number Fields. Math. Comp., Apr. 1976.

Wang, P.; and Minamikawa, T.: Taking Advantage of Zero Entries in the Exact Inverse of Sparse Matrices. Proceedings of the Symposium on Symbolic and Algebraic Computation, Assoc. Comput. Mach., Aug. 1976, pp. 346-350.

Werschulz, Arthur G.: Computational Complexity of One-Step Methods for a Scalar Autonomous Differential Equation. Carnegie-Mellon Univ., Sept. 1976, 21pp.

Werschulz, Arthur G.: Computational Complexity of One-Step Methods for Systems of Differential Equations. Carnegie-Mellon Univ., Sept. 1976, 29pp.

Werschulz, Arthur G.: Optimal Order and Minimal Complexity of One-Step Methods for Initial Value Problems. Carnegie-Mellon Univ., Sept. 1976, 34pp.

Yao, A.C.; and Knuth, D.E.: Analysis of the Subtractive Algorithm for Greatest Common Divisors. Proceedings of the Nat. Acad. Sci., no. 72, 1975, pp. 4720-4722.

Yilmaz, H.: New Approach to Relativity and Gravitation. Annals of Phys., vol. 81, no. 1, Nov. 1973, pp. 179-200.

**Yilmaz, H.:** On the "Derivation" of Einstein's Field Equations. Am. Journ. of Phys., vol. 43, no. 4, Apr. 1975, p. 319.

**Yilmaz, H.:** Generalized Field Theory of Gravitation. Nuovo Cimento, vol. 31 B, no. 8, Feb. 1976, pp. 211-218.

**Yilmaz, H.:** Physical Foundations of the New Theory of Gravitation. Annals of Phys., vol. 101, no. 2, Oct. 1976, pp. 413-432.

**Yilmaz, H.:** A Generalized Nordstrom-Reissner Metric. Lett. Nuovo Cimento, Sept. 1977.

**Yun, D.Y.Y.:** On Algorithms for Solving Systems of Polynomial Equations. SIGSAM Bulletin, Sept. 1973.

**Zippel, R.:** Power Series Expansions in MACSYMA. Proceedings of the Conference on Mathematical Software II, Purdue Univ., May 1974.

**Zippel, R.:** Univariate Power Series Expansions in MACSYMA. Proceedings of Symposium on Symbolic and Algebraic Computation, Assoc. Comput. Mach., Aug. 1976, pp. 198-208.

NASA-Langley, 1977

90

50

END

5.1.78